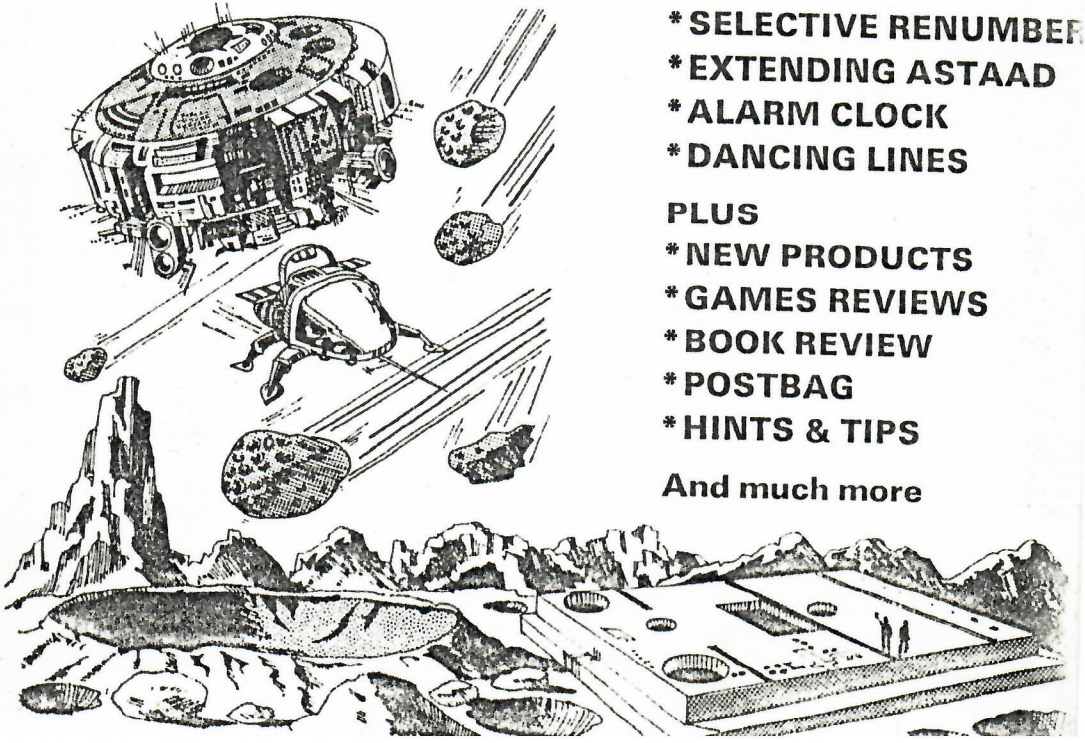
£1.00

ELBUG
ELECTRON

Vol 1 No 6 MAY 1984

LUNAR ESCAPE



GAMES
* **HUNT THE NUMBERS**
* **FOUR IN A ROW**

PLUS
* **SELECTIVE RENUMBER**
* **EXTENDING ASTAAD**
* **ALARM CLOCK**
* **DANCING LINES**

PLUS
* **NEW PRODUCTS**
* **GAMES REVIEWS**
* **BOOK REVIEW**
* **POSTBAG**
* **HINTS & TIPS**

And much more

EDITORIAL

THIS MONTH'S MAGAZINE

Anyone who sets out to do much programming on their Electron will soon realise that the Basic Renumber command is quite limited. This month's utility, a must for all programmers, enables any part of a program to be renumbered, and it checks for any clashes, and any relevant references elsewhere in the program.

Our continuing series on Electron graphics provides some more useful procedures, this time for colouring irregularly shaped areas. All the procedures in this series could form a very useful procedure library in the way we described last month.

In December, we published a program called ASTAAD for computer aided design. We have now extended the features of this program to make it even more impressive. If you are at all interested in any form of computer graphics, then ASTAAD is well worth experimenting with.

We have also picked out three more entertaining games for this issue, ranging from a computer version of 'Connect Four' through to another space game, 'Lunar Escape', all good fun to play.

ELECTRON GRAPHICS

In last month's article in this series, the program called FILLSQUARE2 on page 7 may have resulted in a little confusion. The usual headings for the program became separated from the program itself, and were printed below the accompanying illustration. Sharp-eyed readers will also have spotted the name BEEBUG rather than ELBUG - a slip of the mind on the part of your editor. You can just ignore these extra lines and use the program as correctly listed above the illustration.

ADD-ONS FOR THE ELECTRON

We have referred several times already to the so far disappointing development of the add-on market for Electron users. However, at the Micro User Show at the end of March, several manufacturers were demonstrating add-ons, some of which look very good indeed. We report on these developments elsewhere in this issue. At long last it does seem as though all those desirable extras are beginning to appear.

Mike Williams

TICE BOARD NOTICE BOARD NOTICE BOARD NOTICE BOAI

MAGAZINE CASSETTE

This month sees the launch of our magazine cassette service for ELBUG. Each month all the programs in ELBUG magazine will be available on cassette, starting with this issue. This will save all that wear and tear on the fingers. Cassettes for all previous issues of ELBUG (1 to 5) are also being produced, and all cassettes will be kept in stock. We expect all six cassettes to be ready for dispatch from mid May onwards. Future magazine cassettes will be ready at the time of publication. Full ordering details are on the back cover.

ELBUG MAGAZINE

GENERAL CONTENTS

Page Contents

2 Editorial
4 Hunt the Numbers
6 Postbag
7 Extending ASTAAD
10 Invisible Alarm
11 Selective Renumber Utility
14 New Products at the Micro User Show
15 100 Programs for the Acorn Electr Reviewed
16 Electron Graphics (Part 6)
20 The Latest Computer Games Reviewed
22 Lunar Escape
27 Dancing Lines
29 Using BBC Micro Programs on the Electron (Part 3) Speeding up the Elk
32 Four in a Row

HINTS, TIPS AND INFO

Page Contents

9 Changing the Flash Rates
9 Stepping through Listings
19 Single Key Recovery
26 Space Invader Prompt
26 Preventing the Screen from Scrolling
31 Speed Improvement with Logical Values
31 'NEXT' Effect with LISTO7

PROGRAMS

Page Contents

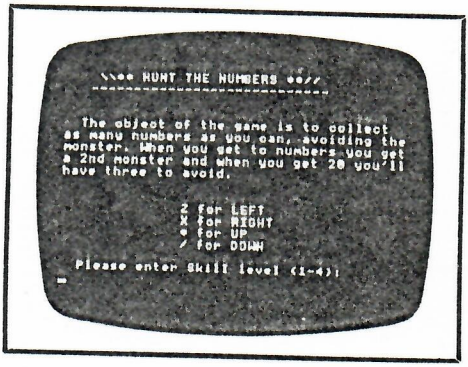
4 Hunt the Numbers Game
7 Extensions to ASTAAD
10 Invisible Alarm
11 Selective Renumber Utility
16 Electron Graphics Examples
22 Lunar Escape Game
27 Dancing Lines Display
32 Four in a Row Game

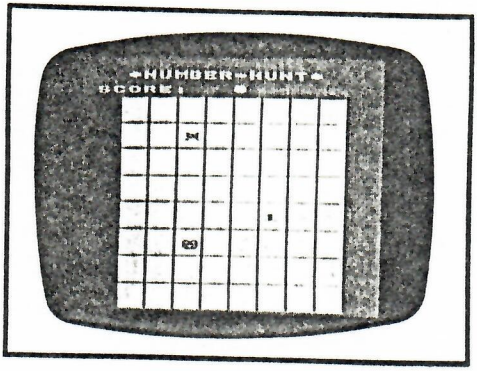
HUNT THE NUMBERS

by K. Allen

Hunt the Numbers is a fast, all-action game written specifically for the Electron. It incorporates coloured graphics in Mode 5 and at the fastest level is very quick and extremely challenging.

Your man is a number-eating beasty, or a kind of number-cruncher, I suppose, whose world consists of a nice bright yellow square. Within this, by the magic of computers, ever increasing numbers keep appearing as temptation to your man to raise his calorific intake for the day. But as you move him to digest these numeric morsels, other nasty monsters will give chase all round the board with only the demise and consumption of your innocent glutton as their objective. These chaps are number-cruncher gulpers and they increase in quantity with your man's success. With higher levels of skill, selected at the beginning of the game, these aggressors become faster and more direct in their descent upon your ever-hungry number-cruncher which can result in a peculiarly loopy dance around the board's surface.


HUNT THE NUMBERS
The object of the game is to collect as many numbers as you can, avoiding the monster. When you get to numbers you get a 2nd monster and when you get 20 you'll have three to avoid.
Z for LEFT
X for RIGHT
* for UP
/ for DOWN
Please enter skill level (1-4):



Controls to change the horizontal and vertical directions of movement are the 'Z','X', and '*' and '/' keys but it's important to note also that simultaneous use of a horizontal and a vertical control key will produce a diagonal motion, which is just as well because the 'nasty monsters' have no qualms about cutting corners when they're chasing you.

```
  10 REM PROGRAM NHUNT
  20 REM VERSION E0.3
  30 REM AUTHOR  K.ALLEN
  40 REM ELBUG   MAY 1984
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 CLEAR:MODE6
 110 ON ERROR GOTO 330
 120 PROCinfo
 130 man%=3:score%=-50:Q%=0
 140 MODE 5
 150 PROCsetup
 160 PROCnumbers
 170 REPEAT
 180 PROCmonster1
 190 PROCman
 200 IF Q%>10:PROCmonster2:ELSE PROCde
lay(6)
 210 PROCman
 220 IF skill%>1:PROCmonster1:ELSE PRO
Cdelay(16)
 230 PROCman
 240 IF skill%>2 PROCmonster1:PROCman:
IF Q%>10:PROCmonster2:ELSE PROCdelay(8)
 250 PROCman
 260 IF skill%>3 PROCmonster1:IF Q%>10
:PROCmonster2:ELSE PROCdelay(8)
 270 PROCman
 280 IF skill%<4:PROCdelay(8)
 290 IF Q%>20:PROCmonster3
 300 UNTIL FALSE
 310 END
```

```
  320 :
  330 ON ERROR OFF:MODE 6:IF ERR<>17 TH
EN REPORT:PRINT "at line";ERL
  340 END
  350 :
 1000 DEFPROCman
 1010 IF Q%>20 AND (A=W AND B=Z) PROCde
ad:GOTO 100
 1020 IF Q%>10 AND (A=F AND B=G) PROCde
ad:GOTO 100
 1030 IF (A=M AND B=N) PROCdead:GOTO100
 1040 IF NOT(A=K% AND B=L%) THEN GOTO 1
050 ELSE VDU 7:PROCnumbers
 1050 IF INKEY(-98):A=A-3:IF A<5 A=5
 1060 IF INKEY(-67):A=A+3:IF A>26:A=26
 1070 IF INKEY(-73):B=B+4:IF B>31:B=31
 1080 IF INKEY(-105):B=B-4:IF B<3:B=3
 1090 IF a=A AND b=B:GOTO 1130
 1100 *FX21,4
 1110 SOUND 0,-15,4,3
 1120 GCOL 0,2:MOVE 40*a,26*b:VDU 241
 1130 GCOL 0,1:MOVE 40*A,26*B:VDU 241
 1140 a=A:b=B
 1150 ENDPROC
 1160 :
 1170 DEFPROCdelay(D%)
 1180 now=TIME:REPEAT UNTIL TIME>now+D%
 1190 ENDPROC
 1200 :
 1210 DEFPROCinfo
 1220 CLS
 1230 VDU 19,1,6;0;
 1240 PRINT TAB(4,3)"\\** HUNT THE NUMB
ERS **//";TAB(3,4) STRING$(28,"-")
 1250 PRINT TAB(2,7)"The object of the
game is to collect  as many numbers as
you can, avoiding themonster. When you
get two digit numbers""you get a 2nd m
onster and when you get""up to 20 you'
ll have three to avoid."
 1260 PRINT TAB(14,14)"Z for LEFT"
 1270 PRINT TAB(14,15)"X for RIGHT"
 1280 PRINT TAB(14,16)"* for UP"
 1290 PRINT TAB(14,17)"/ for DOWN"
 1300 PRINT TAB(2,19)"Please enter Skil
l level (1-4):"
 1310 REPEAT:skill$=GET$:UNTIL INSTR("1
234",skill$)>0
 1320 skill%=VAL(skill$)
 1330 PRINT TAB(33,19);skill%
 1340 PRINT TAB(5,22)"=>PRESS SPACE BAR
 TO START<="TAB(5,23) STRING$(28,"-")
 1350 REPEAT UNTIL INKEY(-99)
 1360 ENDPROC
 1370 :
 1380 DEFPROCsetup
 1390 ENVELOPE 3,2,8,4,8,2,2,2,126,0,0,
-126,126,126
 1400 A=5:B=3:a=5:b=3:f=26:F=26:g=31:G=
31:N=7:n=7:M=26:m=26:W=5:w=5:Z=31:z=31:
K%=0:L%=0
 1410 VDU 24,160;16;1120;848;
 1420 VDU 23,240,102,153,153,255,129,16
5,165,126
 1430 VDU 23,241,195,126,90,90,102,60,6
6,195
 1440 VDU 5:VDU 18,0,130:VDU 12
 1450 VDU 26:VDU 19,0,4;0;:VDU 4
 1460 VDU 19,2,3;0;:VDU 19,1,2;0;
 1470 VDU 19,3,1;0;:VDU 18,0,2
 1480 start=TRUE:start2=TRUE:start3=TRUE
 1490 COLOUR2
 1500 PRINT TAB(1,2);"*HUNT THE NUMBERS
*";TAB(1,4);"SCORE:";TAB(15,4);
 1510 IF man%>1:COLOUR 1:FOR lives=2 TO
 man%:VDU 9,241:NEXT
 1520 VDU 5
 1530 GCOL 0,3
 1540 FOR X=160 TO 1120 STEP 120:MOVE X
,16:DRAW X,848:NEXT
 1550 FOR Y=16 TO 848 STEP 104:MOVE 160
,Y:DRAW 1120,Y:NEXT
 1560 ENDPROC
 1570 :
 1580 DEFPROCmonster1
 1590 IF M=A:GOTO 1600 ELSE M=M+3*(-1-2
*(M<A)):IF M<5 M=5 ELSE IF M>26:M=26
 1600 IF N=B:GOTO 1610 ELSE N=N+4*(-1-2
*(N<B)):IF N<3 N=3 ELSE IF N>31:N=31
 1610 IF (M=F AND N=G) OR (M=W AND N=Z)
:N=n:M=m:ENDPROC
 1620 GCOL 3,1
 1630 MOVE 40*m,26*n:VDU 240:IF start=T
RUE:start=FALSE:GOTO 1630
 1640 MOVE 40*M,26*N:VDU 240
 1650 n=N:m=M
 1660 ENDPROC
 1670 :
 1680 DEFPROCmonster2
 1690 IF F=A:GOTO 1700 ELSE F=F+3*(-1-2
*(F<A)):IF F<5 F=5 ELSE IF F>26:F=26
 1700 IF G=B:GOTO 1710 ELSE G=G+4*(-1-2
*(G<B)):IF G<3 G=3 ELSE IF G>31:G=31
 1710 IF (M=F AND N=G) OR (W=F AND Z=G)
:G=g:F=f:ENDPROC
 1720 GCOL 3,1
 1730 MOVE 40*f,26*g:VDU 240:IF start2=
TRUE:start2=FALSE:GOTO 1730
 1740 MOVE 40*F,26*G:VDU 240
 1750 f=F:g=G
 1760 ENDPROC
 1770 :
 1780 DEFPROCmonster3
 1790 IF W=A GOTO 1800 ELSE W=W+3*(-1-2
*(W<A)):IF W<5 W=5 ELSE IF W>26:W=26
 1800 IF Z=B GOTO 1810 ELSE Z=Z+4*(-1-2
*(Z<B)):IF Z<3 Z=3 ELSE IF Z>31:Z=31
```

```
 1810 IF (W=F AND Z=G) OR (W=M AND Z=N)
:W=w:Z=z:ENDPROC
 1820 GCOL 3,1
 1830 MOVE 40*w,26*z:VDU 240:IF start3=
TRUE:start3=FALSE:GOTO 1830
 1840 MOVE 40*W,26*Z:VDU 240
 1850 z=Z:w=W
 1860 ENDPROC
 1870 :
 1880 DEFPROCnumbers
 1890 MOVE 40*(K%-14.7-(Q%>9)),26*L%:GC
OL 0,2:PRINT Q%
 1900 k%=3*RND(7)+2:l%=4*RND(7)-1:IF k%
=K% AND l%=L%:GOTO1900:ELSE K%=k%:L%=l%
 1910 GCOL 0,0:MOVE 40*(K%-14.7-(Q%>8))
,26*L%:Q%=Q%+1:PRINT Q%
 1920 S=score%:score%=score%+50
 1930 GCOL 0,0:score%=INT(score%):MOVE
64,896:PRINT S:MOVE 64,896:GCOL 0,2:PRI
NT score%
 1940 ENDPROC
 1950 :
 1960 DEFPROCdead
 1970 SOUND 1,3,80,24
 1980 VDU 19,0,0;0;19,1,7;0;
 1990 TIME=0:REPEAT UNTIL TIME=250
 2000 man%=man%-1:IF man%>0 GCOL 0,128:
CLS:PROCsetup:Q%=Q%-1:score%=score%-50:
GOTO 160
 2010 VDU 4:PRINT TAB(2,10)SPC(21)"WOUL
D YOU LIKE ";SPC(23);"ANOTHER GAME?(Y/N
)"SPC(20)
 2020 *FX15
 2030 REPEAT KEY$=GET$:UNTIL INSTR("YyN
n",KEY$)>0
 2040 IF KEY$="N" VDU22,6:END ELSE CLS:
AGAIN=TRUE:ENDPROC
```

POSTBAG POSTBAG POSTBAG POSTBAG POSTBAG POSTBAG

LOCKED OUT

Dear Sir,

As a newcomer to computing I have found ELBUG of great value. However, I have come across something that puzzles me and that is the message 'LOCKED' when loading programs (including the Welcome tape - TURTLE), but I have not found this in the User Guide. This only happens occasionally and I would appreciate your advice.

J.S.Webster

Reply:

This is a feature of cassette files that is not described in the User Guide. Machine code programs can be saved on tape in a 'LOCKED' form so that they may only be *RUN and not be *LOADed. This is used by some software producers as a means of protecting their copyright software. It involves setting just a single bit in the first block of such a program. Trying to *LOAD a locked program will produce the 'LOCKED' message.

Should any other program be poorly recorded, or not otherwise load correctly, the error can sometimes appear to the micro as though you are trying to load a locked file, to which it responds with the message 'LOCKED'. The fact that it only happens occasionally confirms this. If the file really was locked then you would get this message every time you tried to load that program. In this case it seems that either your Welcome tape is a poor recording, or your cassette recorder is not providing a good signal for the micro.

THE MISSING LINK

Dear Sir,

Am I missing out? I enclose two photographs of the Electron circuit board, one from a magazine review and one of my own Electron. The thing that is puzzling me are the two blank spaces on my board labelled IC17 and IC18.

I would be pleased if you could enlighten me on the missing 'bits'.

Robert Hamilton

Reply:

Paul Hulyer is another sharp-eyed reader who has spotted this difference. Nothing is effectively missing from anyone's Electron. Early examples of the Electron, sent out to magazines, including ELBUG, for review contained Basic and the Operating System as two separate 16K chips (IC18 and IC2 to its right). IC17 was needed for the micro to operate in this way. In the majority of production models of the Electron, the Operating System and Basic are both contained within the same 32K chip at IC2, and the chips at the other two locations are now quite redundant.

EXTENDING ASTAAD

by David A. Fell

In the December 1983 issue of ELBUG we presented an excellent program for computer aided design. We said then that the program could be easily extended, and now we have done just that, to provide even more features.

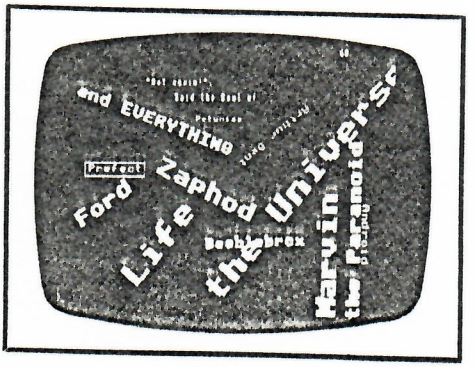
In the original version of ASTAAD, published in ELBUG issue 2, we showed how to extend the program in order to save screens to cassette and how to load them from cassette. We also suggested further enhancements to the graphics facilities of the program.

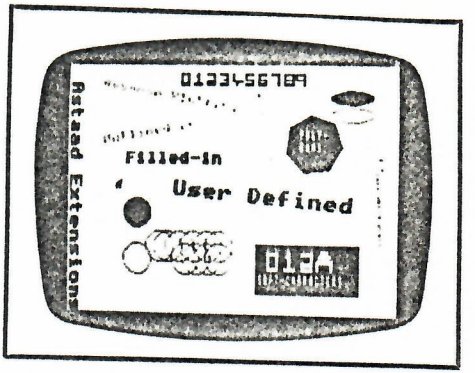
This month, we describe how to add three important new features to improve further the capabilities of this already impressive package. The features added are :

1. The ability to include user-defined characters in 'ASTAAD' text and display them at 'any size, any angle, anywhere on the drawing'.

2. The facility to draw 'filled' circles and polygons as well as outlines only.

3. The facility to reverse foreground and background colours and thus include 'white on black' and 'black on white' text and diagrams in the same drawing.





All the new features are dealt with by control keys rather than function keys as follows.

Ctrl-C Toggle user-defined characters.

Ctrl-F Toggle filled/unfilled polygons and circles.

Ctrl-T Toggle reversal of foreground and background colours.

To use a control key, press the key marked 'CTRL' and the appropriate letter key together. Each time one of these control keys is pressed (or toggled), it changes the state of the feature selected (from 'on' to 'off', or from 'off' to 'on').

USER DEFINED CHARACTERS

'ASTAAD' is also the name given to the routine brought into action by pressing Func-f0. ASTAAD stands for Any Size Text printed at Any Angle on the Drawing. This facility, however, was limited, in the original version, to the normal keyboard characters. An extension has been provided to allow for the soft (or user defined)

characters to be used instead. This allows for new characters to be designed, and then also drawn at any angle, and at any size.

The choice of characters is controlled by Ctrl-C. With this in the 'off' state, as it is when you run the program for the first time, ASTAAD behaves as it did in the old version of the program. When in the 'on' state (selected by pressing Ctrl-C), user defined characters are taken from the keyboard with 'A' selecting character 224, 'B' selecting character 225 and so on. The characters are displayed at the top of the screen as selected. This allows for the new string to be seen as it's being built up. The user defined characters must be set up before running the extended version of ASTAAD (ASTAAD2) with the VDU23 statement (see User Guide pages 93 and 109). If you have already been running ASTAAD2, you may exit from this by pressing Break and typing in OLD and Return. You may now use VDU23 to re-define characters before running ASTAAD2 again, without the need to reload the program from cassette. The character definer on the ELBUG introductory cassette is also an ideal way to design characters that you may like to use in ASTAAD2.

As an example, try typing the following into your Electron:

```
VDU23,224,0,255,191,159,159,145,17,17
```

Now load and run ASTAAD2, and press 'Ctrl-C'.
Press 'Func-f0' for ASTAAD text.
Press 'A' and Return.
Enter a size, say 20, and Return.
Enter an angle, say 10, and Return.

Now watch a primitive elephant being displayed. This illustrates that the soft character is now being read from the memory definition.

FILLED POLYGONS AND CIRCLES

Another new feature, controlled by Ctrl-F, is the in-filling of polygons (and thus also circles, as they are drawn using the polygon routine). Normally, when a polygon is drawn in ASTAAD, it appears only as an outline, but ASTAAD2 allows for either an outline, or a solid polygon to be

ASTAAD
Graphics Design Program

drawn. The selection between the two is made by pressing Ctrl-F. Initially, in ASTAAD2, 'in-fill' is turned off. This means that only the polygon's outline will be drawn. If Ctrl-F is now pressed, then the switch will be altered, and the next polygon to be drawn will be a solid one. The next time the Func-f3 option is used to repeat the last polygon drawn, then whether or not the polygon is filled in is dependent on the current state of the 'in-fill' switch, and not the state of the switch when the initial polygon was drawn. This feature also applies to the circle drawing routine selected by Func-f9.

REVERSING FOREGROUND AND BACKGROUND

When drawing lines, filling in polygons, or any other ASTAAD function, the foreground colour is normally black, and the background colour white (readers may like trying yellow on blue as opposed to black on white by including an appropriate VDU19 at the start of the program - see the User Guide page 107). There is a final switch in ASTAAD2 that affects the drawing colours. To alter this switch, press Ctrl-T. In the 'on' state, just as when the program is first run, the drawing will normally be made in black on a white background. In the 'off' state, the drawing will take place in the reverse colours. This allows for the 'blank out an area' function, invoked by Func-f8, to fill in a rectangular area. Examples of the use of the various functions, including reverse colour, are shown in the illustrations with this article.

EXTENDING THE ASTAAD PROGRAM

The additions to the original ASTAAD program are listed below. They should be typed into the computer AFTER loading in the original ASTAAD, which must not have been renumbered in any

way, since some of the new lines replace lines from the earlier version. Alternatively, save the new section separately using *SPOOL after typing it in, and then use *EXEC to append this to the original ASTAAD (see the article on Procedures and Functions in ELBUG issue 5 for more information on the use of *SPOOL and *EXEC or consult the User Guide, page 200).

We have also included the correct version of line 350 (printed wrongly in the original version) and we suggest the deletion of line 150 if you do not want the audible 'beeping' of the original program.

```
   10 REM Program EXTEND to ASTAAD
   20 REM Version E1.3
   30 REM Author Jim Tonge
   40 REM ELBUG May 1984
   50 REM Program subject to copyright
   60 :
   61 E%=0:Z%=-1:F%=0
   80 ON ERROR:VDU 12,5,26:GOTO 1140
  131 IF key=6 E%=NOTE%:GOTO120
  132 IF key=3 PROCcharacters:GOTO120
  133 IF key=20 PROCcolours:GOTO120
  350 VDU23,1,0;0;0;0;
  520 VDU4,28,6,1,59,1:T$=FNread:INPUT"
Size? (2 to 125):  "S%,"Angle(deg.)?"T%
:CLS:VDU5,26
  550 IF F% PROCread ELSE A%=&BF00+ASC(
MID$(T$,C%,1))*8
  895 IF G%=0 THEN ENDPROC
  910 LOCAL I%,K%,M%,N,C,S,B,D,R,T,Y
  911 IF E% Y=85 ELSE Y=5
  960 IF I%>1 THEN PLOT Y,K%,M% ELSE MO
VE X%,Y%:MOVE K%,M%
  961 IF I%>1 AND E% PLOT85,X%,Y%
  971 IF E% PLOT85,X%,Y%
 1140 IF ERR<>17 THEN ON ERROR OFF:MODE
6:REPORT:PRINT" at line ";ERL:END
 1995 :
 2000 DEF PROCcharacters
 2010 IF F%=0 THEN F%=159 ELSE F%=0
 2020 ENDPROC
 2030 :
 2040 DEF PROCcolours
 2050 Z%=NOT Z%
 2060 IF Z% GCOL0,128:GCOL0,1
 2070 IF NOT Z% THEN GCOL0,0:GCOL0,129
 2080 ENDPROC
 2090 :
 2100 DEF PROCread:LOCALX%,Y%
 2110 Y%=&B:?&B00=ASC(MID$(T$,C%,1))+F%
 2120 A%=10:CALL&FFF1
 2130 A%=&B01
 2140 ENDPROC
 2150 :
 2160 DEF FNread:PRINT''"Text? ";:T$=""
 2170 LOCAL I%:REPEAT I%=GET
 2180 IF I%=21 THEN PRINT STRING$(LEN(T
$),CHR$(127));:T$="":I%=0
 2190 IF I%=127 AND LEN(T$) THEN VDU I%
:I%=0:T$=LEFT$(T$,LENT$-1) ELSE IF I%=1
27 THEN VDU7:I%=0
 2200 IF I% AND I%<>13 THEN T$=T$+CHR$I
%:VDU I%+F%
 2210 UNTIL I%=13:CLS
 2220 =T$
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

CHANGING THE FLASH RATES

The rate at which the flashing colours flash, can be changed using *FX9,m and *FX10,s where m and s are described below.

*FX9,0 forces the first named colour to show continually.

*FX9,m sets the duration for the first colour, with m in 50ths of a second

*FX10,0 forces the second named colour to show continually.

*FX10,s sets the duration for the second named colour, with s in 50ths of a second.

Initially the values of m and s are 25.

STEPPING THROUGH LISTINGS - S.WARD

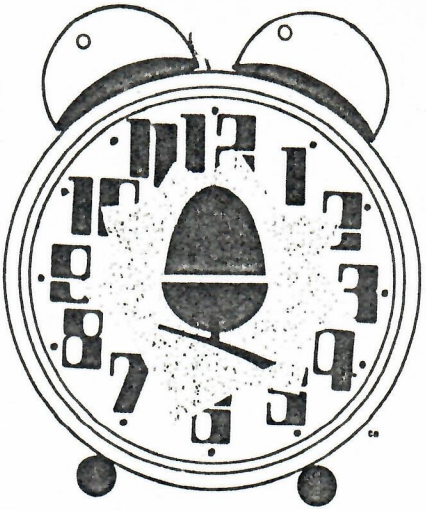
To step through a listing, without it scrolling, enter the following command in immediate mode:

VDU 14 <return>

Every time you list the program, the computer will list a page at a time. To get the next page, press Shift. A similar effect can be obtained by holding down the Control key (marked 'CTRL', and just to the left of the 'A' key) and pressing 'N' (Ctrl-N). The effect can be stopped by entering VDU 15 or pressing Ctrl-O. Listings can also be temporarily halted by pressing Ctrl and Shift together.

INVISIBLE ALARM

by Gordon Weston



How many times have you sat up all night long programming your Electron, when you set out to finish in five minutes? We present here a short routine which will sound an alarm after any preset period of time, up to a year ahead if you wish!

The alarm program will sit invisibly in your machine and produce a beep after a predefined delay. This can be anything from a fraction of a second to several hundred years (though this has not been fully tested yet!). The alarm is not affected by running Basic or machine code programs providing the memory used by this routine is not overwritten or Break is pressed.

When you run the program it asks for the time delay in seconds. Type in the required delay, and then carry on with whatever you want to. When the period of time is up, your Electron will sound an alarm. Unfortunately, we have not been able to provide a 'snooze' button.

TECHNICAL NOTES

The alarm routine has to be in machine code so that it can be set up in an area of memory not used by Basic. In this way you can be using Basic without in any way affecting the alarm set. Memory from address &70 (in hex) up to &8F is reserved for routines like this. Take care when typing in the program, and remember to save a copy before running it.

The program uses a timer which is set initially to a negative value (the number of centi-seconds before the alarm is to sound). Your micro has a built in 'clock' which will keep on increasing the timer until it reaches zero, at which time the alarm routine is called.

The time delay is set as 5 bytes in lines 150 and 160. The counter scans upwards from the negative value given to zero.

Lines 130 and 140 set up a control block in memory (&70 TO &77) which defines the sound parameters to be used when the time limit has been reached. Lines 170 and 180 then point to &007D in memory, where the machine code routine to generate the sound is stored. Line 190 sets up the OSWORD parameters to write the five byte value at &0078 to the interval timer, which is subsequently called in line 200.

It is possible to execute your own section of machine code after the time delay, rather than just generate a beep. This is accomplished by deleting lines 130 and 140 and 210 to 280. Lines 170 and 180 must now point to the low and high bytes, (currently 7D and 0) respectively), of the start address of your code.

```
 10 REM Program ALARM
 20 REM Version E1.0
 30 REM Author G.Weston
 40 REM ELBUG May 1984
 50 REM Program subject to copyright
 60 :
100 INPUT "No. of seconds"n
110 n=n*100
120 *FX14,5
130 !&70=&FFF10001
140 !&74=&001400C8
150 !&78=-n
160 ?&7C=&FF
170 ?&220=&7D
180 ?&221=0
190 A%=4:X%=&78:Y%=0
200 CALL &FFF1
210 P%=&7D
220 [OPT3
230 LDX#&70
240 LDY#0
250 LDA#&7
260 JSR&FFF1
270 RTS
280 ]
290 END
```

SELECTIVE RENUMBER UTILITY

by G. and L. Pettit

This month we present a useful utility program which allows selective renumbering of Basic programs, a much needed aid for Basic programmers.

The RENUMBER command in BBC Basic is limited in its usefulness, as you can only renumber a whole program, and not just part of a program. It is often desirable to number different parts of a program with different groups of line numbers. For example, most programs listed in the magazine place all the procedure definitions starting at line 1000, just to make the layout of the program clearer. Many program standards (including ours for ELBUG magazine) require that each program include initial REMs containing dates, author's name, peripherals required, algorithms employed, etc. Since no two programs will have identical REMs, the main program will start at different line numbers if blanket renumbering is used.

Another instance where selective renumbering is most useful is in using procedure libraries as an addition to a main program (see ELBUG issue 5). These main programs will inevitably be of different lengths and, if the Electron's RENUMBER command is used on the resulting total programs, the line numbers of the procedures will differ from one program to the next. This makes recognition of the procedures difficult.

It is to cater for these and other situations, where selective renumbering is to the programmer's and end-user's advantage, that the Selective Renumber program has been written. It allows renumbering of the leading statements only, or of a portion of the middle of the coding, or of the end statements only. The new starting line number and the new increment (applied to the renumbered lines only) are specified by the user and the program will inform him if there is any overlap between existing line numbers and the new ones BEFORE renumbering takes place. The user may therefore withdraw from an overlap situation and re-specify the parameters, before any confusion occurs.

Although line numbers within REM statements are not altered, line numbers in all other statements will be changed, including calls from the non-renumbered statements. Thus if subroutines are being renumbered, calls from the main program are changed to match, or if only a part of the main program is being renumbered, the GOTO or RESTORE instructions in the rest of the main program will be changed if necessary.

PROGRAM OPERATION

First type in the Selective Renumber program and SAVE on cassette in the usual way. The description that follows assumes that you have saved the Renumber program under the name 'RENUM', though you could of course use any name. To use the program, make sure that you are in Mode 6, load the Basic program to be renumbered, as normal, then type

```
PAGE=TOP+&100   <return>
LOAD"RENUM"     <return>
RUN             <return>
```

This procedure will prevent the Renumber program over-writing the user's program, assuming this is already resident in memory at &E00. The value TOP+&100 has been chosen so that the renumber program is loaded in at the next page in memory, just above the user program, thus giving the maximum space for work.

When Selective Renumber is run, the program will first check that the user's program has been loaded. If not, an error message is displayed and the Renumber program will stop, to allow the user to reset PAGE to &E00 (type PAGE=&E00 <return>), to load his program and to reset PAGE again to TOP+&100 (type PAGE=TOP+&100 <return>). When RUN is typed, a message requests

the range of line numbers to be renumbered. This may be answered by

, nnn for all lines up to nnn,
mmm , nnn for all lines between mmm and nnn inclusive, or
mmm , for all lines from mmm onwards.

The program will next request the new initial line number and new increment, for the section to be renumbered. If the resulting line numbers would conflict with the unchanged numbers, a message will ask the user if he wishes to abandon the renumbering. If he abandons the run, his program will be unaltered, otherwise the renumbering will be carried out without further messages. At the end of the run, the user must return PAGE to its original value by typing PAGE=&E00 <return> before listing his program.

EXPLAINING PAGE

Part of your computer's memory is reserved for its own use. The point where your own program starts is always stored for reference in a special variable called PAGE. On the Electron, this is normally set to &E00 (3584 in decimal), when you first switch the computer on. To check, type PRINT PAGE <return>. PAGE can be set to other values for special purposes, usually so that a program can be located in a different part of memory. It is possible, in this way, to have more than one program in memory at the same time. This happens when the RENUMBER program and your program are in memory together.

Finally, when running the program, please note that it takes an appreciable time to renumber even a small part of a long program. Don't be impatient, just be thankful you're not having to edit all the line numbers on the screen - and go and file your letters or read ELBUG, while your micro does the job for you. You won't use this program every few minutes, but when you get in a tight corner with line numbers, and you can't use the Electron RENUMBER command, this may get you back into production again.

VARIABLES USED

The array E% contains the line numbers of the original program, and the array F% contains the equivalent line numbers in the renumbered portion of the program. Initially, all elements of F% are set to -1 and this is used as an indication of the renumbered range.

N% is PAGE for the program to be renumbered.
L% is the total number of lines in the program.
G% is the first line to be renumbered.
H% is the last line to be renumbered.
D% is the new starting line number of the renumbered part.
M% is the increment of the line numbers in the renumbered part.
P% counts the lines as they are renumbered.

```
   10 REM Program RENUM
   20 REM Authors G & L Pettit
   30 REM Version E1.0
   40 REM ELBUG   MAY 1984
   50 REM PROGRAM SUBJECT TO COPYRIGHT
   60 :
  100 ON ERROR GOTO 230
  110 N%=&E00
  120 MODE6:PRINTTAB(6,2)"SELECTIVE REN
UMBER PROGRAM"
  130 PROClines
  140 DIME%(L%),F%(L%)
  150 PROColdlines
  160 PROCnewlines
  170 PROCtest
  180 PROCline_ren
  190 PROCinfill
  200 PRINT"Time = "TIME DIV100" secs"
  210 END
  220 :
  230 ON ERROR OFF
  240 MODE 6
  250 IF ERR<>17 REPORT:PRINT" at line
";ERL
  260 END
  270 :
 1000 DEFPROClines
 1010 finish=FALSE:L%=0:C%=N%
 1020 REPEAT
 1030 IF ?(C%+1)=&FF finish=TRUE:GOTO10
50
 1040 C%=?(C%+3)+C%:L%=L%+1
 1050 UNTILfinish
 1060 ENDPROC
```

```
 1070 :
 1080 DEFPROColdlines
 1090 C%=N%
 1100 FORI%=1TOL%
 1110 E%(I%)=(?(C%+1))*256+?(C%+2)
 1120 C%=C%+?(C%+3)
 1130 NEXT
 1140 ENDPROC
 1150 :
 1160 DEFPROCnewlines
 1170 FORI%=1TOL%:F%(I%)=-1:NEXT
 1180 N$="***Must be numbers***"
 1190 PRINT'"Give line numbers of first
 and last"'"lines to be renumbered."'"
Use the format e.g.  100,250"
 1200 INPUT" or   ,250 (renumbers up to
 line 250)"'" or 100,   (ditto from lin
e 250 to end)"',G$,H$
 1210 IF ASCG$=-1 G%=0:GOTO1230
 1220 G%=VALG$:IF G%=0 AND G$<>"0"PRINT
'N$:GOTO1190
 1230 IF ASCH$=-1 H%=E%(L%):GOTO1250
 1240 H%=VALH$:IF H%=0PRINT'N$:GOTO1190
 1250 INPUT'"Give new line number for f
irst line and increments (e.g.200,10) "
D$,M$
 1260 D%=VALD$:M%=VALM$:IF D%=0ORM%=0PR
INT'N$:GOTO1250
 1270 P%=0:TIME=0
 1280 PRINT'"Delay for processing"
 1290 FORI%=1TOL%
 1300 IF G%>E%(I%)THENY%=E%(I%):GOTO1340
 1310 IF H%<E%(I%)GOTO1340
 1320 Z%=I%
 1330 F%(I%)=D%+M%*P%:P%=P%+1
 1340 NEXT
 1350 P%=D%+M%*(P%-1)
 1360 ENDPROC
 1370 :
 1380 DEFPROCtest
 1390 clash=FALSE
 1400 FORI%=1TOL%
 1410 Q%=F%(I%):IF Q%=-1GOTO1460
 1420 FORJ%=1TOL%
 1430 IF Q%<E%(J%) J%=L%:GOTO1450
 1440 IF Q%=E%(J%) PROCclash
 1450 NEXT
 1460 NEXT
 1470 IF (Z%+1)>L% Z%=L%-1
 1480 IF D%<=Y% PRINT'"Your ranges over
lap!":clash=TRUE
 1490 IF (Z%+1)>L% OR D%>E%(L%) GOTO1510
 1500 IF P%>=E%(Z%+1) PRINT'"Your range
s overlap!":clash=TRUE
 1510 IF clash INPUT'"Do you want to go
 on (Y or N) "A$:IF A$<>"Y"ANDA$<>"y" E
ND
 1520 ENDPROC
 1530 :
 1540 DEFPROCclash
 1550 IF Q%<G% OR Q%>H% PRINT"Clash in
line ";Q%:clash=TRUE
 1560 ENDPROC
 1570 :
 1580 DEFPROCline_ren
 1590 C%=N%
 1600 FORI%=1TOL%:F%=F%(I%)
 1610 IF F%=-1GOTO1640
 1620 ?(C%+1)=F%DIV256
 1630 ?(C%+2)=F%MOD256
 1640 C%=C%+?(C%+3)
 1650 NEXT
 1660 ENDPROC
 1670 :
 1680 DEFPROCinfill
 1690 finish=FALSE
 1700 C%=N%
 1710 REPEAT
 1720 B%=?C%:IF B%=&20 C%=C%+1:GOTO1720
 1730 IF B%=13GOTO1790
 1740 IF B%=&22 REPEAT C%=C%+1:B%=?C%:U
NTILB%=&22:C%=C%+1:GOTO1720
 1750 IF B%<&8B GOTO1770
 1760 IF B%=&8B OR B%=&8C OR B%=&E4 OR
B%=&E5 OR B%=&F7 PROCsub
 1770 C%=C%+1:GOTO1810
 1780 PRINT~R%,~S%,~T%
 1790 IF ?(C%+1)=&FF finish=TRUE:GOTO18
10
 1800 C%=C%+4
 1810 UNTIL finish
 1820 ENDPROC
 1830 :
 1840 DEFPROCsub
 1850 C%=C%+1:B%=?C%:IF B%=&20 OR B%=&E
5 GOTO1850
 1860 IF B%<>&8DGOTO2060
 1870 C%=C%+1:U%=0
 1880 R%=?C%:S%=?(C%+1):T%=?(C%+2)
 1890 S%=S%-&40
 1900 T%=(T%-&40)*256
 1910 IF R%MOD&10=0 U%=16384:R%=R%+&4
 1920 IF R%=&54 R%=0:GOTO1960
 1930 IF R%=&44 R%=64:GOTO1960
 1940 IF R%=&74 R%=128:GOTO1960
 1950 IF R%=&64 R%=192
 1960 O%=U%+R%+S%+T%
 1970 FORI%=1TOL%
 1980 IF E%(I%)<>O% GOTO2020
 1990 IF F%(I%)=-1 GOTO2010
 2000 PROCinsert
 2010 I%=L%
 2020 NEXT
 2030 C%=C%+2
 2040 B%=?(C%+1):IF B%=&20 C%=C%+1:GOTO
2040
 2050 IF B%=&2C C%=C%+1:GOTO1850
 2060 ENDPROC
 2070 :
 2080 DEFPROCinsert
```

```
2090 U%=F%(I%)
2100 V%=U% DIV&4000
2110 W%=U% MOD&4000
2120 S%=W% MOD64+&40
2130 T%=W% DIV256+&40
2140 U%=W% MOD256
2150 W%=U% DIV64
2160 IF W%=1 W%=-&10:GOTO2190
2170 IF W%=2 W%=&20:GOTO2190
2180 IF W%=3 W%=&10
2190 R%=W%+&54-4*V%
2200 ?C%=R%:?(C%+1)=S%:?(C%+2)=T%
2210 ENDPROC
```

NEW PRODUCTS AT THE MICRO USER SHOW

by Nigel Harris

At the last Micro-User exhibition at the end of March it was evident that firms already involved in the manufacture of add-ons for the Electron's 'Big Brother' computer have more than just an eye out for a similarly developing Electron market. Spurred on by the belief that the Electron will be as successful as the BBC micro, a number of devices are beginning to appear that will extend the Electron's facilities towards that of the Beeb. This will greatly enhance the capability of this very attractive micro, and allow a much wider range of software to be used.

DISC SYSTEMS

Pace were showing early versions of their disc interface for the Electron. Due for release around June, it plugs on to the edge-connector at the back of the Electron. This of course, would allow you to load and save programs on floppy disc much more quickly than cassette, very useful to anyone who continually finds themselves waiting for their cassette recorder. Floppy disc drives are quite complex devices, and require direct computer control. A complete disc system would normally therefore, come complete with a disc filing system program to control the interface between the computer and the disc machine, and render many disc transactions invisible to the user. The Pace disc interface will be supplied with their own 'DFS' which, they say is functionally equivalent to their 'AMCOM DFS' already available for the BBC micro. (We shall look at discs in more detail in a later issue of ELBUG).

JOYSTICKS

Two firms, First Byte Computers and Signpoint Ltd, had on show interface boxes to allow the use of joysticks on the Electron (inclusive prices £24.95 and £16.95 respectively). Both support the Atari type of switching joystick. There are a large number of joysticks to this standard already available. However, as many BBC owners have found out, there are two functionally different types of joystick around. One is the switched type that these interfaces cater for, which is quite suitable for games use. The other is the continuously variable type that allows the input of infinitely many intermediate values if required to the computer, and is often used for more serious graphics applications (this is the analogue type of system that is standard on the BBC micro). Like the disc interface, these two joystick interfaces will plug into the back of the Electron, but neither extends the expansion interface to allow for other add-ons. They must come at the end of one that has been expanded, or other add-ons to the Electron unplugged.

Addresses of suppliers:

Pace Disc Systems, 92 New Cross St., Bradford BD5 8BS.

First Byte Computing, 10 Castlefields, Main Centre, Derby DE1 2PE.

Signpoint Ltd., 166a Glyn Road, London E5.

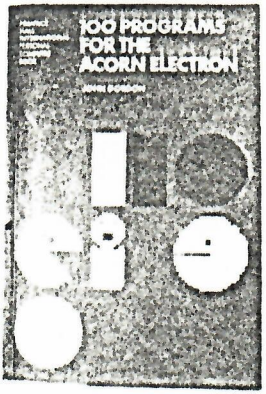
[We hope to review the two joystick interfaces more fully in the next issue of ELBUG.]

100 PROGRAMS FOR THE ACORN ELECTRON

Reviewed by Mike Williams

This book, and cassette of the same title, can be purchased together or separately. 100 programs is a lot of computing in either format but is it value for money? Mike Williams looks at this well produced package and reports.

Title: 100 Programs for the Acorn Electron
Publisher: Prentice Hall International
Price: £6.95 (book)
£11.50 (cassette)



The book contains a listing of each program which is clearly printed (unlike many computing books), with many instructions indented to clarify the structure of the program. Each program is accompanied by brief notes on the use and design of the program in question. The binding of the book made it very difficult (in fact impossible) to keep the book open at any listing, so that typing in any program was very much a one handed and somewhat difficult task.

The cassette is well packaged to look like the book, and comes complete with a 32 page booklet containing notes on each program, very similar to those in the book. The only omission here is the book's list of contents which conveniently classifies the programs under various headings,and a general introductory chapter.

With 100 programs on offer, clearly the emphasis is on quantity, rather than quality,and consequently many of the programs are quite short. They cover a wide range including games (of course), business and home applications, mathematics and science applications and graphics examples. I believe that there are too many mathematical examples for the typical Electron user, and in contrast, there is insufficient coverage of string and text handling. This perhaps reflects the author's academic background, but even so, current ideas on the teaching of programming have long since moved on from the idea of the computer as a number machine.

The program descriptions, and indeed even the instructions for their use in some cases, are often frustratingly short in the information provided. Many of the programs should really be considered as basic routines which the reader could integrate into a more polished program. As such, this collection could form an invaluable resource, particularly for the novice programmer. In view of this, I feel it is unfortunate that the notes accompanying many of them are so short, with often inadequate explanation of how the program works.

Despite some of the criticisms that I have made above, I am sure that many Electron owners, interested in learning more of programming, will find much of interest in these programs. Either book or cassette on their own would be sufficient, the former giving the advantage of printed listings for reference, the latter the convenience of direct cassette loading. Few people will quibble at programs costing a mere 7p (book) or 11.5p (cassette) each, though I do believe that half the number of programs and twice the explanation would have provided a more useful publication. But then, I suppose, the title wouldn't have been quite so eye-catching!

ELECTRON GRAPHICS (Part 6)

by Mike Williams

This month in our series on Electron Graphics we look at some of the more interesting and also amusing effects that can be achieved using the PLOT command.

Last month we looked at how to use the PLOT85 command to produce a variety of filled in shapes, but all based on the triangle. It would be rather nice if Basic allowed us to fill in and colour any shape, and indeed BBC Basic does just that in another variation of the PLOT command.

To fill in an area of literally any shape on the screen is in fact quite a complex task, and although this can be achieved just using Basic, we shall concentrate on some much simpler but never-the-less very useful fill routines. Sophisticated fill routines are included in commercially available graphics packages such as the 'Electron Graphics System' from Salamander Software that we reviewed in ELBUG issue 4, and in PAINTBOX shortly to be released by us for the Electron.

The fill instruction that we shall be using takes the form:

```
PLOT77,x,y
```

where the '77' specifies which PLOT option we are going to use, and 'x,y' refers to a point on the screen in the usual graphics co-ordinates. This instruction draws a line in the current drawing colour to the right and to the left of the point given. The line is continued in both directions until a point is reached which is not in the background colour, or the edge of the screen is reached.

Now this may well sound rather complicated, and you might also be wondering how this can be used to fill in an area, since the instruction just draws a straight line. Let's look at a very simple example:

```
100 MODE 2
110 GCOL 0,3
120 PLOT77,640,512
130 END
```

We shall be using Mode 2 in all the examples this month as it allows us the choice of all 16 colours. Line 110 selects yellow as the drawing colour, and the background is black by default. The PLOT instruction in line 120 starts at the point (640,512), the centre of the graphics screen, and draws a yellow line horizontally right and left till it reaches the edges of the screen.

At this point it will be useful to introduce a procedure to draw an outline circle in white, and of any radius and in any position on the screen. This is very similar to the basic circle routine described in part 4 of this series in ELBUG issue 4. The main difference is the inclusion of a GCOL command to ensure that the circle is drawn in white, while VDU29 is used to reset the graphics origin back to (0,0) after moving it to the centre of the circle at the start of the procedure. Here is the procedure:

```
1000 DEF PROCcircle(radius,x,y)
1010 LOCAL angle,X,Y
1020 VDU29,x;y;:GCOL 0,7
1030 MOVE radius,0
1040 FOR angle=0 TO 2*PI STEP PI/16
1050 X=radius*COS(angle)
1060 Y=radius*SIN(angle)
1070 DRAW X,Y
1080 NEXT angle
1090 VDU29,0;0;
1100 ENDPROC
```

Assuming that you have typed this procedure into your Electron, add this variation on our first program given above:

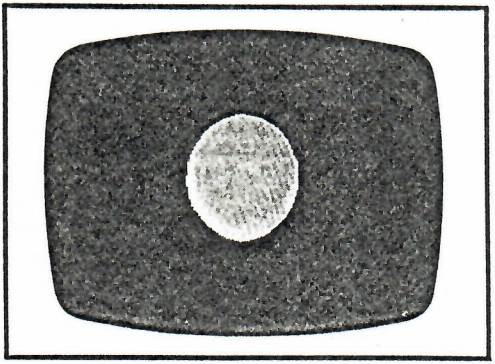
```
100 MODE 2
110 PROCcircle(200,640,512)
120 GCOL 0,3
130 PLOT77,640,512
140 END
```

This very short program draws a white circle of radius 200 and with its centre at (640,512). The PLOT command

in line 40 draws a yellow line as before, but it stops as soon as it reaches the circumference of the circle because this is no longer the background colour.

You might like to try some variations on this program, by changing the x and y co-ordinates given in the PLOT command. You should find that as long as you choose a point inside the circle, you get a horizontal yellow line inside the circle. If you choose a point that is outside the circle, then the line will reach at one end to the circumference of the circle, and at the other end to the edge of the screen (or indeed to both edges of the screen). The usefulness of this PLOT instruction lies in the fact that we do not need to know in advance where the ends of the line will be.

All we need to do now to fill completely the inside of the circle is to repeat the PLOT77 command from the bottom to the top (or top to bottom if



you wish). Yes, you guessed it - we need to use a FOR-NEXT loop. Rewrite your main program as follows:

```
100 MODE 2
110 PROCcircle(200,640,512)
120 GCOL 0,3
130 FOR Y%=312 TO 712
140 PLOT77,640,Y%
150 NEXT Y%
160 END
```

If you run this program you should find that the white circle is neatly filled with a yellow centre (of course!).

Now although this seems to work quite well, it is very easy indeed to improve the speed. To see how, we need to refer back to a previous article in this series, that in ELBUG issue 4, where we looked briefly at screen resolution.

Mode 2 that we have been using provides low resolution graphics, and the screen actually consists of 160 points horizontally by 256 points vertically, compared with the graphics co-ordinates that we use of 1280 by 1024. What this means is that in the vertical direction, for example, we have to move by a minimum of 4 graphics units in order to move 1 physical point on the screen. If we move in steps of less than 4 then we are still referring to the same physical screen point.

This means that in our Mode 2 programs any vertical movement in steps of less than 4 is wasted, and in the horizontal direction we need steps of at least 8 to move physically on the screen. If we return to our previous program you should see that we can change line 130 to read:

```
130 FOR Y%=312 TO 712 STEP 4
```

If you run the revised program, you should find that you get exactly the same result as before, but four times faster!

Similar factors apply in all graphics modes. The minimum vertical step size is always 4 (physically 256 points on the screen), but the minimum horizontal step size varies according to the mode: 2 in Mode 0, 4 in Modes 1 and 4, and 8 in Modes 2 and 5.

We have now seen enough to write a more general fill routine, that will fill not just a circle, but any shape, though with some limitations. We will write a procedure that will specify a point on the screen, and will fill the surrounding area. We will assume that the existing colour on the screen at the point specified is the background colour, and that the boundaries of the area to be filled will be marked by lines or areas of a different colour, or the edges of the screen, whichever are reached first.

In filling the circle we moved in one direction only, from bottom to top. With a more general approach, we no longer know where the boundaries are going to occur, and so we need to fill in the area in two sections, both starting from the point specified. The first section will be to fill in line by line moving vertically upwards from the starting point, and then to fill downwards from the starting point. In both cases we continue moving and filling until we encounter a point which is not in the background colour.

The processes of first filling up from a point, and then filling down from the point are essentially the same, so this has been written as a procedure called PROCfillupdown, which is then called twice in the main procedure, PROCfill. These two procedures are defined as follows:

```
1200 DEF PROCfill(colour,x,y)
1220 PROCfillupdown(colour,x,y,4)
1230 PROCfillupdown(colour,x,y-4,-4)
1240 ENDPROC
1250 :
1300 DEF PROCfillupdown(colour,x,y,inc)
1310 newy=y:background=POINT(x,y)
1320 GCOL0,colour:GCOL0,128+background
1330 REPEAT
1340 PLOT77,x,newy
1350 newy=newy+inc
1360 UNTIL POINT(x,newy)<>background
1370 ENDPROC
```

Let's look first at the procedure PROCfillupdown, as this is the one which does all the work. Notice the POINT instruction. This tells us the colour (as a number) of any point on the screen. The parameters x,y represent the starting point for filling an area, so the first task is to find the colour of this point and set it as the background colour using the GCOL command (lines 1310, 1320). Remember that background colours are always specified by adding 128 to the colour number. The other two parameters of this procedure are the colour to use for filling (colour) and the increment (inc) for filling up (4) or down (-4).

Since we no longer know in advance the start and end points we cannot readily use a FOR-NEXT loop this time, but this is exactly the situation for using REPEAT-UNTIL. We shall repeatedly call the PLOT77 command, as we move up or down the screen, until the colour of the next starting point is different to the background colour, again detected by using the POINT function. Because we are using REPEAT-UNTIL we must ourselves program the incrementing of the new value of y (newy) to move up (or down) the screen.

The fill procedure itself simply calls the other procedure twice, once to fill up from the starting point, and once to fill downwards from the starting point.

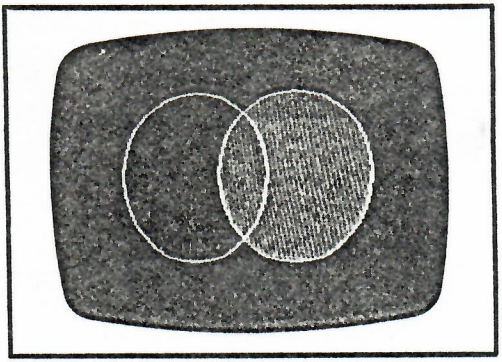
So now we have a general fill procedure that will enable us to fill in (or colour) any area, starting from any point on the screen. Some examples will soon expose the limitations of this procedure and you will begin to see why a really general fill routine is bound to be quite complex.

Our procedure will fill any circle provided the starting point is directly between the highest and lowest points of the the circle. Any other position, and the filling will stop as soon as the starting point for each line reaches the circumference. You will also have to be careful about using this fill procedure for concave areas, or you may well find 'holes' being left unfilled.

Despite these limitations, this procedure does have a lot of uses, some of them quite amusing as we shall see in the two examples with which we shall finish this month's article. Both of these examples require all three procedures that we have already been using. The first example draws two overlapping circles on the screen, and then colours in each of the three enclosed areas with a different colour. Any problems are avoided by carefully selecting a central starting point for each area. Here is this example:

```
100 MODE 2
110 ON ERROR GOTO 510
120 PROCcircle(300,440,512)
130 PROCcircle(300,840,512)
140 PROCfill(1,440,512)
```

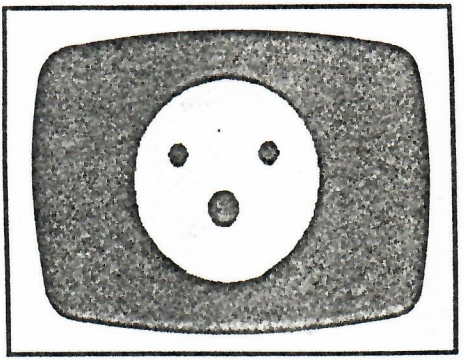
```
150 PROCfill(2,640,512)
160 PROCfill(4,840,512)
190 END
```



The final example has to be run to see the full effect. The program first draws a large circle and colours it in. Two small circles are then added and coloured as eyes, and a third circle added and coloured as a nose. A REPEAT-UNTIL loop is then used to repeatedly open and close one eye to give the effect of winking. This is done by colouring in the eye with the same colour as the rest of the face, and then again with the colour of the eye. The starting points are carefully selected just inside the top and bottom edges in each case to give the realistic effect of an eye first closing and then opening again.

```
100 MODE 2
110 ON ERROR GOTO 510
120 PROCcircle(400,640,512)
125 PROCfill(5,640,512)
130 PROCcircle(50,440,612)
140 PROCcircle(50,840,612)
150 PROCfill(4,440,612)
160 PROCfill(4,840,612)
170 PROCcircle(75,640,400)
180 PROCfill(1,640,400)
190 REPEAT
200 PROCfill(5,440,658)
210 PROCfill(4,440,568)
220 Z=INKEY(200)
230 UNTIL FALSE
240 END
```

I hope that you will be able to think of further ways to experiment with this fill routine. Although a fill routine



may seem attractive, it can be be complex, and the triangle plotting described last month will often be simpler to use, particularly when the shape of an area to be coloured is known in advance. Next month we will continue to look at more simple variations in the use of the PLOT command.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SINGLE KEY RECOVERY - C.Luck

The following single function key definition will attempt to recover as much of a bad program as possible, enabling it to be listed, and corrupt lines corrected. When you need it, press function key zero (Func-f0), and the computer will display the lines successfully recovered.

```
*KEY0MODE6:p=&E00:?p=13:p?1=0:REPEAT:c=4:REPEAT:c=c+1:UNTILp?c=13:p?3=c:P.p?1*256+p?
2:p=p+c:UNTILp?1=&FF|M
```

This is much more limited than the Rescue program in issue 2 of ELBUG but might be well worth trying in the first instance.

THE LATEST GAMES REVIEWED

Title :737 Flight Simulator
Supplier:Salamander Software
Price :£9.95
Reviewer:Mike Williams
Rating :****

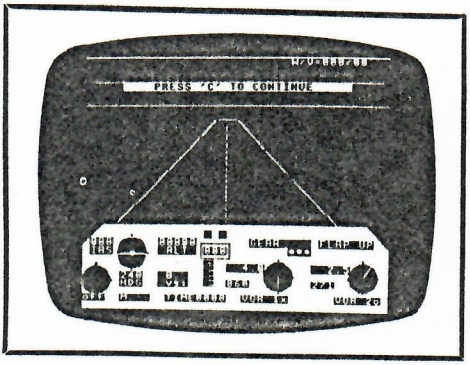
If you have ever fancied your chances piloting a modern jet airliner, then this program gives you that opportunity, in this computer simulation. You are the pilot of a Boeing 737 jet with a full display of all the necessary instruments on the screen in front of you. The top part of the screen either shows a visual impression of the runway during take-off and landing, or a radar plot of the runway approach and the immediate vicinity.

This is certainly a comprehensive and realistic simulation. The cockpit display adequately shows 22 different instruments, making good use of the Electron's graphics facilities. A further clue to your status is provided by the reassuring sound of the engines' turbine whine in the background. This amount of feedback is very useful for the 'pilot', as there are 27 keyboard commands to control all aspects of the plane! Fortunately there is also a very good manual, which not only lists in detail all the instruments and controls, but also gives you the necessary information for take-off and landing.

Take-off is not too difficult - you just use maximum thrust to achieve sufficient take-off speed and then climb (make sure you raise the landing gear and retract the flaps quickly though). You can then climb the aircraft up to any height you like, banking and setting course as you wish. You'll soon find that you need to brush up on your navigation as the plane flies out of radar range and you have to rely solely on the instruments.

Assuming you are able to find your way back to the airport, the most challenging task of all is to make a safe landing. I found I needed to practice this several times before I was able to land the plane successfully, and even more to do so regularly. The program helps by allowing you to practice just landings (as well as take-offs), and in the event of a crash (very likely) allows you to take a metaphorical step back and try again (a pity you can't do this with the real thing!).

I found this a fascinating and exciting program to use; one which is rather different from the normal run of computer games, and with many more features than there is space to describe here. Even though I can now take off and land reliably, after a flight of up to half an hour, I don't think I'm quite ready for the real thing. My only slight criticism is that I personally lost interest once I could take-off and land without coming too close to crashing the plane, although there are other features that could maintain your interest longer (such as setting up a strong crosswind or specifying your own airport layout). Despite the small criticism, I thoroughly recommend this program.



GAMES WITH A 'BYTE'
FROM ALLIGATA SOFTWARE
Reviewed by Alan Webster

Title : BLAGGER
Price : £7.95
Rating : ****

Blagger is one of three new games for the Electron from Alligata Software. It has twenty (yes twenty!) different screens, each with its own title and level of complexity. It will take you a long time to get through the first few screens, but persevere - it's well worth the effort.

The object of the game is to retrieve the keys to a safe. Once all the keys on that level have been found, you can open the safe and proceed to the next level. There are a number of lifts and coveyor belts that have to be negotiated, as well as a host of different creatures to avoid. The game is very simple to operate, needing only three keys, and is an excellent example of its genre.

Title : GUARDIAN
Price : £7.95
Rating : *****

In my view, Guardian is the ultimate game for the Electron. This game is a version of the arcade game called Defender, and it does run very well. Although speed limits the game to a four colour mode, the game is still fast and very smooth, with some of the best sound effects you are likely to find on an Electron.

At one time, it was thought that a version of Defender for the Electron would be impossible because of the sheer speed necessary, the sideways scrolling and the extent of the graphics and movement involved. Alligata Software have not only proved this view wrong, but provided the fastest game I have so far seen for the Electron.

The idea of the game is to navigate your spacecraft over the surface of a planet, and to shoot the marauding aliens before they abduct the humananoids. If the aliens reach the top of the screen with a humanoid, they mutate and hunt you down with unbelievable ferocity. Additional features include other undesirables such as baiters, pods and swarmers. An extra ship is awarded at every 10000 points.

Like all good games, this one requires a degree of practice before you appreciate how exceptional it is. A high score of 658,000 for Guardian was achieved by the author, Steven Evans, at the recent Micro User Show. However, if you don't quite measure up to this level, still send your high scores into us for inclusion in our high score table for the Electron.

Title : BUG BLASTER
Price : £7.95
Rating : ***

Bug Blaster is a very good implementation of the arcade game called Centipede. The action takes place in a mushroom patch full of centipedes, spiders, frenzied fleas, and a mushroom poisoning scorpion called 'Brian'.

Your main objective is to wipe out the centipedes before they take over. The job is not too easy, especially in the later stages when the game speeds up and the action becomes fast and furious, although the early stages do tend to become rather boring after a while.

All of these games include a high score table and extra life features. They are available direct from Alligata Software Limited, 178 West Street, Sheffield S1 4ET.

LUNAR ESCAPE

by C. J. Hall

The following game, written entirely in Basic, is an implementation of the arcade game Lunar Rescue, where you control a space shuttle ferrying survivors from the lunar surface to the orbiting space ship. The game is quite fast, is fun to play and very addictive.

In this game, you have to steer a rescue shuttle safely through a belt of asteroids, touching down on one of the three landing pads where you take on board a survivor. You then launch the craft away from the pad and steer your way back through the asteroids, finally docking safely with the main ship. The survivor disembarks, and the process is repeated. Each landing gets harder since the landing pads are partially burnt away by the shuttle's thrusters during each launch, leaving a smaller landing pad for the next trip.

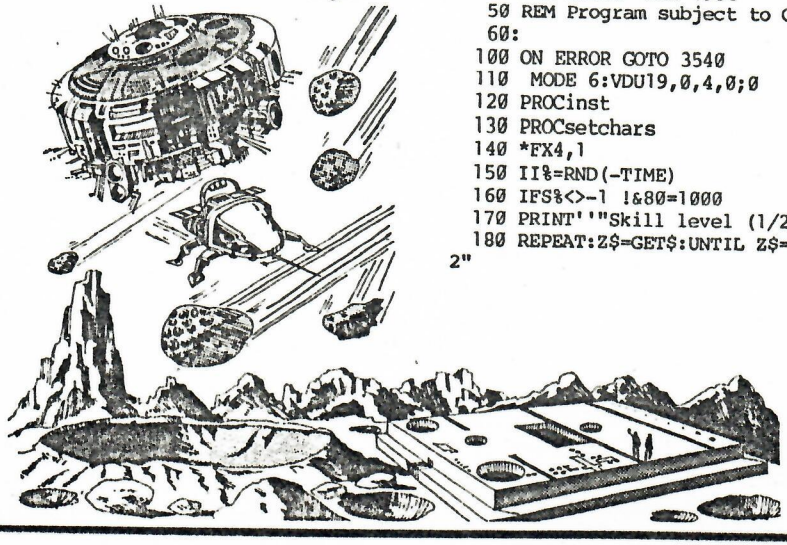
The four controls for the shuttle are as follows:

Space bar - releases shuttle from main craft or launches it from the landing pad.
Return - brakes as the shuttles climbs or descends.
Z - moves the shuttle left.
X - moves the shuttle right.

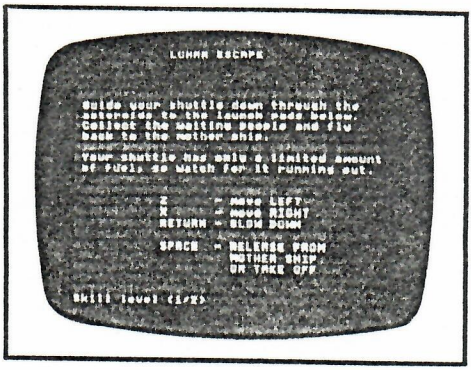
You have a choice of two skill levels, level 1 having fewer asteroids than level 2. On successive screens the number of asteroids increases and they also move faster as well as diagonally, making the game progressively harder.

The listing is quite long and complicated, so take great care when entering the program as debugging could be difficult due to the complicated interactions between the various parts of the program.

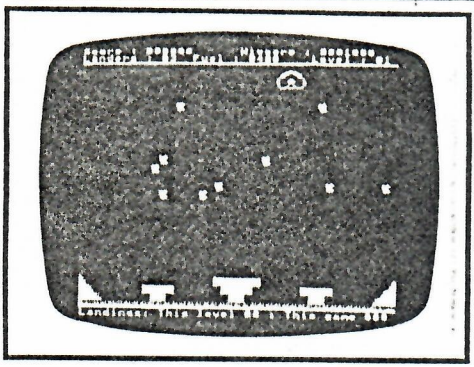
This is a deceptively simple game, which is much harder to play successfully than it might appear initially. Like all good computer games this one is quite addictive, and well worth the effort of typing it in.



```
  10 REM Program LUNAR
  20 REM Version E0.2
  30 REM Author  C.J.Hall
  40 REM ELBUG  MAY 1984
  50 REM Program subject to Copyright
  60:
 100 ON ERROR GOTO 3540
 110  MODE 6:VDU19,0,4,0;0
 120 PROCinst
 130 PROCsetchars
 140 *FX4,1
 150 II%=RND(-TIME)
 160 IFS%<>-1 !&80=1000
 170 PRINT''"Skill level (1/2)"
 180 REPEAT:Z$=GET$:UNTIL Z$="1"ORZ$="
2"
```


LUNAR ESCAPE
Guide your shuttle down through the asteroids to the launch pads below. Collect the waiting people and fly back to the mother ship.
Your shuttle has only a limited amount of fuels so watch for it running out.
Z - move LEFT
X - move RIGHT
RETURN - SLOW DOWN
SPACE - to RELEASE FROM MOTHER SHIP OR TAKE OFF
Skill level (1-5)

```
  190 IF Z$="1" G%=FALSE:AT%=5 ELSE G%=
TRUE:AT%=10
  200 MODE4:VDU23,1,0;0;0;0;:VDU 19,0,4
;0;
  210 *FX15,0
  220 DOCK=20:LL%=1:S%=0:B%=17000:I%=B%
:FI%=400:F%=FI%:X%=0:NS%=FALSE:U%=0:FLA
G1%=FALSE
  230 NL%=3:IN%=FALSE:AN%=FALSE:D%=0:FT
%=TRUE:A%=981434236:SP$=CHR$(32)
  240 W%=16777183:Y%=-66822:Y1%=-842972
9:H%=!&80:V%=FALSE:BBB%=V%:*FX11,0
  250 CLS:DIMX%(41),Y%(41),IX%(41),IY%(
41):NG%=0
  260 L$=STRING$(40,CHR$238):PRINTTAB(0
,0)"Score : 000000"
  270 PRINTTAB(0,2);L$;:PRINTTAB(20,0)"
Hiscore : 000000"
  280 PRINTTAB(0,1)"Landers : 00";TAB(14
,1)"Fuel : 0000";TAB(29,1)"Level : 00";
  290 PRINTTAB(0,31)"Landings: This lev
el 00 : This game 000";
  300 SEA1$=STRING$(40,CHR$253)
  310 STMP$=STRING$(2,SP$)+CHR$247+CHR$
245+CHR$249+STRING$(2,SP$)+CHR$10+STRIN
G$(7,CHR$8)+SP$+CHR$243+SP$
  320 STM2$=SP$+CHR$244+SP$+CHR$10+STRI
NG$(7,CHR$8)+STRING$(2,SP$)+STRING$(3,C
HR$248)+SP$+SP$
  330 SHIP$=STMP$+SP$+STM2$
  340 Q$=SHIP$:ANY$=STRING$(3,CHR$232)
  350 WHT$=STRING$(2,CHR$234)
  360 FORII%=0TOH%STEP(H%DIV100):PRINTT
AB(37-LEN(STR$(II%)),0);II%
  370 SOUND0,-15,3,1:NEXT
  380 PRINTTAB(37-LEN(STR$(H%)),0);H%
  390 SOUND1,-15,150,4
  400 FORII%=1 TO 3:PRINTTAB(11,1);II%:
SOUND0,-15,3,3:NEXT
  410 FORII%=20TOF%STEP20:PRINTTAB(25-L
EN(STR$(II%)),1);II%:SOUND0,-15,7,1:NEXT
  420 SOUND1,-15,150,8:PRINTTAB(38,1)"1"
  430 MANSHIP$=STMP$+CHR$(255)+STM2$
  440 ENTER$=STMP$+SP$+SP$+CHR$244+SP$
  450 PRINTTAB(0,30);SEA1$;
  460 PRINTTAB(0,29);:VDU234,234,232:PR
INTSPC(34);:VDU233,234,234
  470 PRINTTAB(0,28);:VDU234,232:PRINTS
PC(36);:VDU233,234
  480 PRINTTAB(0,27);CHR$232;SPC(38);CH
R$233;
  490 PRINTTAB(9,29);WHT$;SPC(8);WHT$;S
PC(8);WHT$;
  500 PRINTTAB(8,28);WHT$;WHT$;SPC(6);W
HT$;WHT$;SPC(6);WHT$;WHT$;
  510 PRINTTAB(17,27);WHT$;WHT$;WHT$;
  520 IF IN%=TRUE GOTO 1860
  530 L%=RND(25)+5
  540 M%=3
  550 PRINTTAB(L%,M%);SHIP$
  560 N%=RND(3)-2:IF N%=0 GOTO 560
  570 J%=0:K%=0
  580 TIME=0
  590 PROCsetastros
  600 PROCdispastros
  610 BBB%=FALSE
  620 REPEAT
  630 PROCmaninship
  640 PROCmoveastros
  650 UNTIL V%=TRUE
  660 REPEAT
  670 PROCmoveman
  680 PROCmoveastros
  690 PROCmoveship
  700 UNTIL BBB% OR U%=1
  710 IF BBB% THEN BBB%=FALSE:GOTO 620
  720 X%=0
  730 REPEAT
  740 PROCmoveastros
  750 PROCmoveship
  760 UNTIL INKEY(-99)
  770 PRINTTAB(QW%,K%+1)STRING$(6,SP$)
  780 REPEAT
  790 PROCmoveastros
  800 PROCupwego
  810 PROCmoveship
  820 UNTIL BBB% OR DOCK<6:IF BBB% THEN
 BBB%=FALSE:GOTO 620
  830 PROChavewedocked
  840 *FX15,0
  850 GOTO 620
  860 END
  870:
 1000 DEFPROCmoveship:T%=L%+N%:IFT%=0 O
R T%=35 N%=-N%:T%=L%+N%
 1010 L%=T%:PRINTTAB(L%,M%);SHIP$:ENDPR
OC
 1020:
 1030 DEFPROCmaninship:T%=L%+N%:IFT%=0
OR T%=35 N%=-N%:T%=L%+N%
 1040 L%=T%:PRINTTAB(L%,M%);MANSHIP$:IF
 INKEY$(2)<>SP$ ENDPROC
 1050 V%=TRUE:PRINTTAB(L%,M%+2);SPC10:P
RINTTAB(L%+3,M%+1);SP$
```



```
 1060 SOUND1,-15,220,1:J%=L%+3:K%=M%+3
 1070 PRINTTAB(J%,K%);CHR$(255):PRINTTA
B(L%,M%);SHIP$:X%=0:ENDPROC
 1080:
 1090 DEFPROCsetastros
 1100 FOR Q%=0 TO AT%+1
 1110 X%(Q%)=RND(37)+1
 1120 Y%(Q%)=RND(18)+6
 1130 IX%(Q%)=RND(5)-3:IF IX%(Q%)=0 GOT
O 1130
 1140 IF G%=FALSE GOTO 1160
 1150 IY%(Q%)=RND(3)-2
 1160 IF Q%MOD5=0 PROCmoveship
 1170 NEXT
 1180 ENDPROC
 1190:
 1200 DEFPROCmoveastros
 1210 IF AT%<10 BG%=1:E%=AT%:GOTO 1240
 1220 IF FT%=TRUE BG%=1:E%=AT% DIV 2:FT
%=FALSE:GOTO 1240
 1230 BG%=AT% DIV 2:E%=AT%:FT%=TRUE
 1240 FOR Z%=BG% TO E%
 1250 T%=X%(Z%)+IX%(Z%)
 1260 R%=Y%(Z%)+IY%(Z%)
 1270 IF T%<0 PRINTTAB(X%(Z%),Y%(Z%))SP
$;:T%=39:X%(Z%)=T%:GOTO1290
 1280 IF T%>39 PRINTTAB(X%(Z%),Y%(Z%))S
P$;:T%=0:X%(Z%)=T%
 1290 IF R%<7 PRINTTAB(X%(Z%),Y%(Z%))SP
$:R%=24:Y%(Z%)=R%:GOTO1330
 1300 IF R%>24 PRINTTAB(X%(Z%),Y%(Z%))S
P$:R%=7:Y%(Z%)=R%:GOTO1330
 1310 PRINTTAB(X%(Z%),Y%(Z%))SP$;
 1320 Y%(Z%)=Y%(Z%)+IY%(Z%)
 1330 X%(Z%)=X%(Z%)+IX%(Z%)
 1340 PRINTTAB(X%(Z%),Y%(Z%));CHR$(254);
 1350 NEXT
 1360 P%=!((K%*40*8)+4+&5800+(J%*8))
 1370 IF P%=A% BBB%=TRUE:PROClosealife
 1380 ENDPROC
 1390:
 1400 DEFPROCdispastros
 1410 FOR II%=1 TO AT%:PRINTTAB(X%(II%)
,Y%(II%));CHR$(254);:NEXT:ENDPROC
 1420:
 1430 DEFPROCupwego
 1440 X%=0
 1450 IF INKEY(-98) X%=-1
 1460 IF INKEY(-67) X%=1
 1470 IF INKEY(-74) PROCwaitabit
 1480 IF AN%=TRUE AN%=FALSE:ENDPROC
 1490 PROClessenfuelbyone:J%=J%+X%
 1500 IF BBB%=TRUE THEN ENDPROC
 1510 IF J%<0 J%=0:X%=0
 1520 IF J%>39 J%=39:X%=0
 1530 IF K%<6 DOCK=5:ENDPROC
 1540 IF K%<9 SHIP$=ENTER$:PRINTTAB(L%,
M%+2);SPC(7)
 1550 K%=K%-1
 1560 PRINTTAB(J%-X%,K%+1)SP$;
 1570 P%=!((K%*40*8)+4+&5800+(J%*8))
 1580 PRINTTAB(J%,K%);CHR$(255);
 1590 IF P%=0 ENDPROC
 1600 PROClosealife:BBB%=TRUE:ENDPROC
 1610:
 1620 DEFPROChavewedocked
 1630 PRINTTAB(L%,M%+2)SPC6
 1640 IF J%<L%+2 OR J%>L%+4 YES=FALSE E
LSE YES=TRUE
 1650 IF YES=FALSE J%=J%-X%:PROCloseali
fe:BBB%=TRUE:ENDPROC
 1660 PRINTTAB(L%,M%);MANSHIP$
 1670 SHIP$=Q$
 1680 J%=L%+3:K%=M%+1
 1690 IF NS%=FALSE oldfuel%=F%:F%=F%+10
0:IF F%>FI% THEN F%=FI%
 1700 IF NS%=FALSE PROCdispfuel(oldfuel
%)
 1710 IF NS%=TRUE NS%=FALSE:PROCnewscre
en
 1720 V%=FALSE:U%=0:DOCK=20
 1730 SHIP$=Q$
 1740 ENDPROC
 1750:
 1760 DEFPROCnewscreen
 1770 IN%=TRUE
 1780 D%=0
 1790 PROCclearastros
 1800 PRINTTAB(0,12)"Well done ... you
cleared a screen !!"
 1810 PROCnoise:PROCnoise
 1820 PROCbonuscount
 1830 PRINTTAB(0,16)"You have a bonus of
 ":PRINTTAB(20,16);B%;:PRINT" points !"
 1840 PRINTTAB(21,31)"00";
 1850 GOTO450
 1860 PROCaddbonus
 1870 AT%=AT%+5
 1880 IF AT% MOD 2=1 AT%=AT%+5
 1890 IF AT%>40 AT%=40
 1900 IF AT%>9 BG%=0:E%=AT% DIV 2
 1910 PROCsetastros
 1920 PROCclearastros
 1930 PROCdispastros
```

```
 1940 LL%=LL%+1
 1950 IF LL%=3 NL%=NL%+1:PRINTTAB(12-LE
N(STR$(NL%)),1);NL%:PROCnoise
 1960 SOUND1,-15,150,26
 1970 PRINTTAB(39-LEN(STR$(LL%)),1);LL%
 1980 I%=I%+3000:B%=I%:TIME=0
 1990 FI%=FI%+100
 2000 oldfuel%=F%:F%=FI%
 2010 PROCdispfuel(oldfuel%)
 2020 I%=I%+4000
 2030 B%=I%
 2040 D%=0
 2050 ENDPROC
 2060:
 2070 DEFPROCclearastros:VDU 28,0,24,39
,7:CLS:VDU 28,0,31,39,0:ENDPROC
 2080:
 2090 DEFPROCdockingssofar
 2100 PRINTTAB(39-LEN(STR$(TD%)),31);TD
%;:PRINTTAB(22,31);D%;:ENDPROC
 2110:
 2120 DEFPROCmoveman
 2130 X%=0
 2140 IF INKEY(-98) X%=-1
 2150 IF INKEY(-67) X%=1
 2160 IF INKEY(-74) PROCwaitabit
 2170 IF AN%=TRUE AN%=FALSE:ENDPROC
 2180 J%=J%+X%:PROClessenfuelbyone
 2190 IF BBB%=TRUE THEN ENDPROC
 2200 IF J%<0 J%=0:X%=0
 2210 IF J%>39 J%=39:X%=0
 2220 K%=K%+1
 2230 PRINTTAB(J%-X%,K%-1)SP$;
 2240 P%=!((K%*40*8)+4+&5800+(J%*8))
 2250 PRINTTAB(J%,K%);CHR$(255);
 2260 IF P%=0 ENDPROC
 2270 IF P%=A% OR P%=W% PROClosealife:B
BB%=TRUE:ENDPROC
 2280 IF P%=Y% OR P%=Y1% FLAG1%=2+(P%=Y
%):PROClosealife:BBB%=TRUE:ENDPROC
 2290 PROCblock
 2300 U%=1
 2310 ENDPROC
 2320:
 2330 DEFPROCblock
 2340 IF J%<13 QW%=6
 2350 IF J%<26 AND J%>12 QW%=17
 2360 IF J%<39 AND J%>25 QW%=26
 2370 PRINTTAB(J%,K%-1);CHR$(255)
 2380 PRINTTAB(J%,K%);CHR$234;
 2390 IF K%=28 SC%=100 ELSE IF K%=29 SC
%=200 ELSE SC%=50
 2400 D%=D%+1
 2410 IF D%=7 NS%=TRUE
 2420 TD%=TD%+1
 2430 PROCdockingssofar
 2440 PROCnoise
 2450 S%=S%+SC%:PROCscore
 2460 K%=K%-1
 2470 ENDPROC
 2480:
 2490 DEFPROClosealife
 2500 SHIP$=Q$
 2510 NL%=NL%-1
 2520 PRINTTAB(12-LEN(STR$(NL%)),1);NL%
 2530 PRINTTAB(J%,K%);CHR$242
 2540 PROCcrash
 2550 PRINTTAB(J%,K%)SP$
 2560 IF FLAG1% THEN PRINTTAB(J%,K%);CH
R$(231+FLAG1%);:FLAG1%=FALSE
 2570  IF NL%=0 PROCfinish:IF Z$<>"Y" E
ND
 2580 V%=FALSE:U%=0:DOCK=20
 2590 PRINTTAB(0,30);SEA1$;
 2600 J%=L%+3:K%=M%+2
 2610 *FX15,0
 2620 IF NS%=FALSE ENDPROC
 2630 PROCnewscreen:DOCKS=0:NS%=FALSE:E
NDPROC
 2640:
 2650 DEFPROCfinish
 2660 VDU28,0,31,39,3:CLS:*FX15,0
 2670 PRINTTAB(9,12)"G A M E   O V E R"
 2680 PRINTTAB(0,25)"Do you want to pla
y again (Y/N)":VDU28,0,31,39,0
 2690 IF S%>H% PRINTTAB(0,17)"You got t
he high-score : well done!!":H%=S%:!&80
=H%
 2700 S%=-1:Z$=GET$:IF Z$<>"N" RUN
 2710 *FX4
 2720 VDU22,6:*FX11,40
 2730 ENDPROC
 2740:
 2750 DEFPROCbonuscount
 2760 B%=I%-TIME:B%=B% DIV 10:IF B%<0 B
%=0:ENDPROC ELSE ENDPROC
 2770:
 2780 DEFPROCaddbonus:S%=S%+B%:FOR II%=
S%-B% TO S% STEP 10
 2790 PRINTTAB(14-LEN(STR$(II%)),0);II%
:SOUND0,-15,3,1:NEXT
 2800 PRINTTAB(14-LEN(STR$(S%)),0);S%:P
ROCnoise:ENDPROC
 2810:
 2820 DEFPROClessenfuelbyone
 2830 F%=F%-1
 2840 IF F%=-1 F%=FI%:PROClosealife:PRO
Cdispfuel(10):BBB%=TRUE
 2850 PRINTTAB(21,1)"0000"
 2860 PRINTTAB(25-LEN(STR$(F%)),1);F%;
 2870 ENDPROC
 2880:
 2890 DEFPROCdispfuel(oldfuel%):PRINTTA
B(21,1)"0000"
 2900 FOR II%=oldfuel% TO F% STEP 10:PR
INTTAB(25-LEN(STR$(II%)),1);II%
 2910 SOUND0,-15,3,3:NEXT:ENDPROC
 2920:
 2930 DEFPROCwaitabit
 2940 IF F%<11 AN%=FALSE:ENDPROC
```

DANCING LINES

by David A. Fell

This intriguing program not only produces an interesting graphics display, but also shows how a sequence of random numbers generated in Basic is not really random at all!

This short program produces an attractive screen display by simulating the bouncing of two points around the screen at random, with a randomly coloured line drawn between the two. An apparently random sequence is built up on the screen, and then erased in the same order as it was created. Once the screen has 'emptied', the program repeats, producing a different sequence for each screen.

RANDOM NUMBERS

The two key facts that enable the program to precisely erase the lines, without any co-ordinates being stored, are the Exclusive-OR plotting technique used (see the User Guide page 97), and the repeatability of Basic's random number generator. This is not a true random number generator, but produces what are termed pseudo-random numbers. This means that the numbers that it produces may appear random, and indeed will satisfy most tests for randomness, but are actually produced by a fixed set of calculations which can be made to repeat. Whenever a 'seed', or starting point, for the Electron's random number generator is entered, it is possible to identically regenerate a particular series of 'random' numbers.

To seed the random number generator, you use the form:

```
X=RND(N)
```

where N is a negative number. As an example, try this:

```
PRINT RND(-100)
```

When 'seeding' the Electron's random number generator, the value that RND returns is the same as was passed to it, in this case -100. Once a negative value has been passed, the generator is 'seeded', and thus now predictable. Moreover, if you now type:

```
PRINT RND(10)
```

for example, immediately after seeding the generator as above you will always get a 9 printed, which is always the first value to be returned in the particular sequence chosen.

PROGRAM NOTES

In the program, line 150 creates a random seed, and this is used at lines 160 and 180 to start the generator off producing the same sequence of numbers. Lines 170 and 190 call the procedure to produce a screen of lines, and both the calling and seeding sections are contained within a REPEAT ... UNTIL loop that cycles endlessly, or until Escape is pressed.

At the start of the program, line 120 turns off the cursor, as this only serves as a distraction in graphical displays. Line 130 effectively turns off the flashing colours, which are inevitably going to be generated as a result of both the usage of Exclusive-OR plotting and the random colour range that is selected, and which would appear unattractive if left flashing. This is achieved by setting the 'mark' period to zero, and thus leaving all of the flashing colours permanently in only one of their two possible colours.

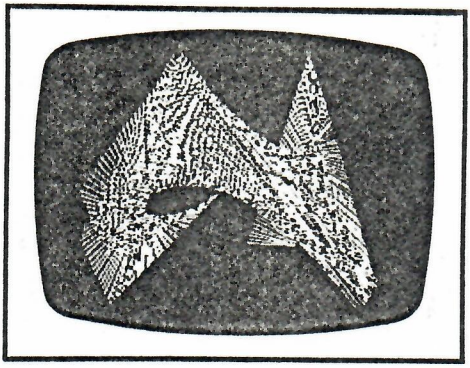
FLASHING COLOURS

All the flashing colours show each of two colours alternately on the screen. The time for which each of the two colours is displayed can be altered by using the *FX9 call to set the time for the first colour and *FX10 to set the time for the second (see page 280 of the User Guide). Setting either time to zero (but not both) results in the other colour being permanently displayed.

The display itself is produced by the procedure PROCpicture. Lines 1010 to 1080 generate random starting points and random increments for the two ends

of the line. The increments are fixed so that they will start the two ends moving off in opposing directions. The FOR-NEXT loop starting at line 1090 will determine how many lines will be drawn for each pattern, and adjusting the value of 500 to, say, 100 will give a shorter pattern.

The procedure uses the Exclusive-OR plotting option of the GCOL instruction (selected at line 1100), and the random colour. Exclusive-OR plotting has the

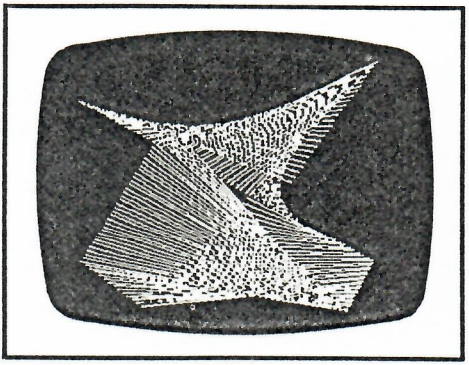


unusual property in that, if any graphics with the same colour and co-ordinates are plotted an even number of times, then they 'cancel' out, leaving whatever happened to be on the screen before. As we start of with a blank screen, and all plotting is done twice, at the same co-ordinates and in the same colour sequence, the net result is that there is nothing left on the screen by the end of the sequence.

The overall effect, therefore, is of a completely random pattern in shape and in colour, and yet one which can be precisely repeated, and hence erased from the screen.

Lines 1090 to 1250 are concerned with the mechanics of bouncing a point around an enclosed rectangle (in this case the screen).

```
 10 REM Program Dancing Lines
 20 REM Author David A. Fell
 30 REM Version E1.2
 40 REM ELBUG MAY 1984
 50 REM Program subject to Copyright.
 60 :
100 ON ERROR GOTO 230
110 MODE 2
120 VDU23,1,0;0;0;0;
130 *FX9
140 REPEAT
150 SEED%=ABS(RND)
160 A%=RND(-SEED%)
170 PROCpicture
180 A%=RND(-SEED%)
190 PROCpicture
200 UNTIL FALSE
210 END
220 :
230 ON ERROR OFF: MODE 6
240 IF ERR<>17 THEN REPORT:PRINT" at
line ";ERL
250 END
260 :
1000 DEF PROCpicture
1010 X1%=RND(1279)
1020 Y1%=RND(1023)
1030 X2%=RND(1279)
1040 Y2%=RND(1023)
1050 XI1%=RND(4)*5
1060 YI1%=RND(4)*5
1070 XI2%=-RND(4)*5
1080 YI2%=-RND(4)*5
1090 FORI%=0TO500
1100 GCOL3,RND(16)-1
1110 MOVE X1%,Y1%
1120 DRAW X2%,Y2%
```



```
1130 IF X1%+XI1%>1279 XI1%=-RND(4)*5
1140 IF X1%+XI1%<0 XI1%=RND(4)*5
1150 IF Y1%+YI1%>1023 YI1%=-RND(4)*5
1160 IF Y1%+YI1%<0 YI1%=RND(4)*5
1170 IF X2%+XI2%>1279 XI2%=-RND(4)*5
1180 IF X2%+XI2%<0 XI2%=RND(4)*5
1190 IF Y2%+YI2%>1023 YI2%=-RND(4)*5
1200 IF Y2%+YI2%<0 YI2%=RND(4)*5
1210 X1%=X1%+XI1%
1220 Y1%=Y1%+YI1%
1230 X2%=X2%+XI2%
1240 Y2%=Y2%+YI2%
1250 NEXT
1260 ENDPROC
```

USING BBC MICRO PROGRAMS ON AN ELECTRON (Part 3)

by David Graham

As I suggested in the first article in this series, one of the major differences between the BBC Micro and the Electron is its speed of operation. The Electron runs at up to three times slower than the Beeb. This can cause serious problems when adapting BBC Micro software for the Electron - as anyone at Acornsoft, Program Power or Beebugsoft will tell you! But there are ways around the problem, and this month, in the last article of this series, I want to look at how to speed up Basic programs on the Electron.

MODE DEPENDENCE

The most striking thing about the speed of the Electron when compared to the BBC Micro, is that the Elk's speed, unlike that of the Beeb, is Mode dependent. If you run the program "Time Test" you will get some idea of this. The program simply prints out the time taken by the Electron to calculate the sine of an angle two hundred times over. Set your Electron to Mode 6, and run it. Note the result printed on the screen, then run it in Mode 2. You will see that it takes more than twice as long. The exact relative speeds of the different Modes depends on the program that you are running - but the speed difference between Modes 0, 1 or 2 and 4, 5 or 6 will always be significant.

```
10 REM TIME TEST
20 TIME=0
30 FOR A%=1 TO 200
40 X=SIN(12)
50 NEXT
60 PRINT TIME
```

From this we can easily establish certain rules of thumb. These are as follows:

1. If speed is important in a program, keep to as high a screen Mode as possible.

2. Perform as many of the time critical parts of your program while in a high Mode. For example, if you are generating a maze for a game in Mode 2, make sure that you are in Mode 6 while you calculate the maze. Only switch to Mode 2 at the last moment.

CHANGING MODE

In most situations where a time critical BBC micro program uses Modes 0, 1 or 2 (the slowest on the Electron), you can change these to Mode 4 or 5 - see table 1. There are several provisos here. If you replace Mode 0 by Mode 4 you will get 40 rather than 80 column text, and you will also lose resolution. Replacing Mode 1 with Mode 4, or Mode 2 with Mode 5 you will not lose resolution, but the number of available colours will be reduced from 4 to 2, or from 16 to 4 respectively.

Mode	Chrs	Cols	Resol-ution	Replace with Mode
0	80	2	640x256	4
1	40	4	320x256	4
2	20	16	160x256	5
3	80	2	-	-
4	40	2	320x256	-
5	20	4	160x256	-
6	40	2	-	-

One of the most commonly used Modes for games on the BBC micro - and where consequently speed is of a premium - is Mode 2. Fortunately it is usually possible to replace this with Mode 5. Character size and resolution are identical in both Modes - and the only difference is the loss of colours. Mode 5 leaves you with only four, including the background. Fortunately four colours is usually just about enough to get by with. We have adopted this technique for example with the game Hedgehog (ELBUG Vol. 1 No. 1). This was originally published in BEEBUG as a

Mode 2 BBC micro game. The way to approach the problem is simply to look for all Mode statements in your program - the Utility Editor published in ELBUG No. 5 might be helpful here. Then just change all occurrences of Mode 2 for 5, and Modes 0 and 1 for 4. There is no real need to change Mode 7 to Mode 6 since the Electron gets into Mode 6 whenever it encounters the instruction "MODE 7".

Now run the program and see what happens. You will almost certainly need to alter some of the new colours produced. One reason for this is that some things printed on the screen will probably have become invisible. What has really happened is that they have been printed in the same colour as the background. The way to sort out the colours is to find every COLOUR or GCOL statement - again the Utility Editor could be helpful here - and ensure that the colour number called is in the range 0 to 3. You will have to choose the numbers carefully so that objects show up as well as possible.

Remember though, that you are not stuck with the four colours black, yellow, red, and white. Each of these may be changed for any of the 16 available colours (but you can still only have a total of four different colours on the screen at any one time). The way to change colours is by using the VDU 19 call. See the User Guide page 107. Suppose that you have converted a Mode 2 screen to Mode 5, and have assigned colours 0, 1, 2 and 3 in such a way as to keep all parts of the screen distinguished, but wish to change all the red objects to cyan. Just insert the statement VDU 19,1,6,0,0,0 after the last Mode change - and so on.

OTHER STRATEGIES

I want to look now at a series of further strategies for increasing the speed of programs adapted from the Beeb - although like those treated above, the principles involved will in general be equally effective if you are originating the code yourself - and wish to make your program as 'fast' as possible. Some of the principles treated here are touched upon in appendix 'E' of the User Guide which gives notes on 'Fast and efficient programs'.

TIME DELAYS

One of the most obvious things to look for in a Beeb program that you are trying to speed up are time delays. The most likely forms of delay are as follows:

```
For A=1 to 100:NEXT
or
TIME=0:REPEAT UNTIL TIME = 20
or
X=INKEY(20)
```

The first two are very obviously waiting loops, and can be removed to test their effect. You have to be more careful with the third type, because as well as performing a waiting function it also assigns a value to X. The easiest way to test its effect is to remove the (20) or whatever, and see what effect this has on the program.

INTEGER VARIABLES

The next thing to do is to replace as many variables as possible by integer variables - and where possible single letter integers. For example, suppose your program has the following variables:

```
INVADER$
LOOPCOUNT
LASERX
LASERY
BULLET%
SCORE
TEXT$
```

INVADER$ and TEXT$ are string variables (the $ tells you this), and should not be changed. BULLET% is an integer variable (the % tells you this). You could change this for B%, but there will not be much saving here. Bigger savings are achieved when you change any of the others to integer variables. You could just call the A%, B%, C% etc. and change each occurrence throughout the program, but unfortunately you cannot always substitute integers for so-called 'real' variables.

The problem with integers is that they can only take a certain range of values: they can never represent decimal numbers. But looking at the

reals that we have - ie. LOOPCOUNT, LASERX, LASERY and SCORE - they may well all be used as integers. Loop counts are often integers - you can't go round a loop 1.5 times - but do check that the step size is also integer or the program will not work correctly when the loop count is converted to integer. The same goes for the position of the laser base in a game, or the score. In this case you could replace all occurrences of LOOPCOUNT by say C%, of LASERX by X%, of LASERY by Y% and of SCORE by S%. This is likely to improve speed considerably.

If possible you should also replace all normal division signs "/" by "DIV", the much quicker integer division function - but again you will lose decimal precision. The answer to 5 DIV 4 is 1, not 1.25. Very often in games, where speed is of the essence, the extra precision is not necessary.

REMS AND PROGRAM LINES

In Acorn's advice on making programs run faster they suggest that you remove all REM statements, and reduce the number of program lines. For example the four lines below:

```
100 REM CALCULATION SECTION
110 X%=100
120 Y%=50
130 IF Z%*SINX>40 THEN P%=50
```

could be rewritten as:

```
100 X%=100:Y%=50:
IFZ%*SINX>40THENP%=50
```

and so on. The speed gained here will not be great however, though every little helps and readability will be reduced.

The User Guide also suggests that you use REPEAT....UNTIL loops. REPEAT loops are certainly faster than unstructured GOTO loops; but what the Guide does not say is that FOR...NEXT loops are the fastest of all - and by a considerable margin. So try to use these where at all possible - and as the Guide does mention, leave off any possible variable in the NEXT statement. Thus use NEXT rather than say NEXT X - the X is quite redundant, and slows execution.

By using a combination of these techniques you will be able to considerably enhance the speed of Electron programs; and if you are converting programs from the Beeb you will find that most can be converted satisfactorily if these various points are born in mind. There will always be some that cannot be satisfactorily converted from a speed point of view however, and there is a certain degree of skill in spotting which these are before you have invested too much time and effort on them.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SPEED IMPROVEMENT WITH LOGICAL VALUES - R.Jefferyes

Internally TRUE and FALSE are held as integers, represented by -1 and 0 respectively. So, for example, if you write

```
crash=TRUE
```

the TRUE will be converted to floating point form when it is stored. The statement will execute faster if you use an integer variable like this:

```
crash%=TRUE
```

'NEXT' EFFECT WITH LISTO7 - M.Robinson

You may know that FOR..NEXT loops can be terminated with commas e.g. NEXT,, (which has the same effect as NEXT:NEXT:NEXT). However these commas are not recognised by the LISTO7 option, used to format listings by indenting by two spaces for each nested loop. Loops terminated by a comma will not be indented by two extra spaces for each NEXT that has been replaced by a comma.

This also applies if the controlling variable of the loop is included e.g. NEXT A,B,C. The actual keyword 'NEXT' must be included to cancel the indentation e.g. NEXT:NEXT:NEXT.

FOUR IN A ROW

by J. Webb

This is an excellent and colourful implementation of the game popularly known as 'Connect Four'. In this game of strategy, you try to arrange four counters in a line in any direction, while preventing your opponent from doing the same.

This is a game for one or two players, with the option of choosing the computer as the opponent. The two players take turns deciding in which column to place a counter. As in the real game, counters always end up in the lowest empty position in any column chosen. The object of the game is to arrange a row of four of your own counters, either horizontally, vertically or diagonally, on the 6 x 7 position board, while trying to prevent your opponent from doing the same. The winner is the first person to get four counters in a row.

The program displays instructions for playing the game on the screen. You can choose whether to play against the computer or another person before the game starts. You then simply select which column to place your counter in by typing in the column number followed by Return.

The usual red and yellow colours are used in the program for the counters, but the yellow can be a little difficult to see against the surrounding white frame in certain circumstances. To change it to magenta (colour 5), for example, add the following line:

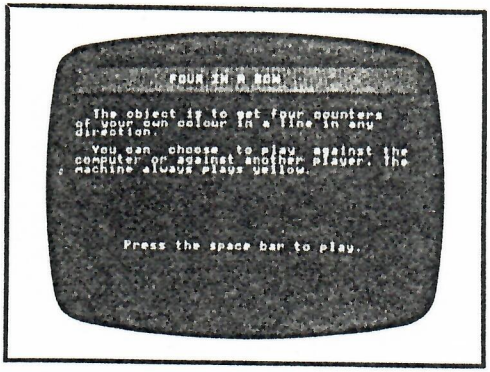
```
245 VDU19,2,5,0,0,0
```

You may also like to try alternative colours.

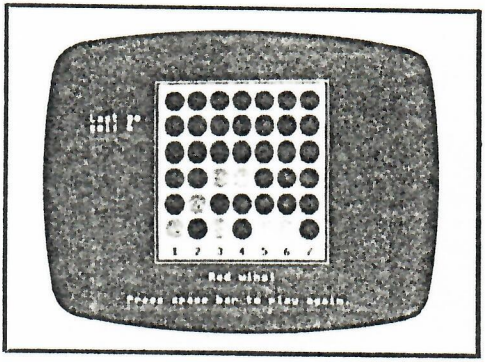
You will find that the computer plays a reasonably strong game and generally speaking, it will win unless you can set a trap for it by gradually building up two interrelated rows of four counters.

```
  10 REM Program FOUR IN A ROW
  20 REM Version E1.0
  30 REM Author J.Webb
  40 REM ELBUG May 1984
  50 REM Program subject to Copyright
  60 :
 100 MODE 1
 110 ON ERROR GOTO 340
 120 DIMposition%(6,7):PROCcircles
 130 ENVELOPE1,6,0,0,0,0,0,0,0,121,
-10,-5,-2,120,120
 140 ENVELOPE2,6,4,-8,-4,16,16,32,
64,64,-64,-64,128,0
 150 PROCtitles
 160 go%=1:win%=0:turn%=0:comp%=0
 170 FORrow%=1TO6
 180 FORcol%=1TO7
 190 position%(row%,col%)=0
 200 NEXT:NEXT
 210 CLS
 220 PROCsetupgame
 230 REPEAT
 240 PROCturn
 250 IF go%=0 AND comp%=0 PROCcolour
 260 IF go%=0 AND comp%=1 PROCcomputer
:PROCcolour
 270 IF go%=1 PROCcolour
 280 PRINTTAB(0,8)"Last go"'"col: ";co
lumn%
 290 SOUND&11,1,86,111:screen%=14347+4
8*column%:REPEAT:screen%=screen%+1920:U
NTIL?screen%>0
 300 missweigh%=0:PROCweighandcheck:mi
ssweigh%=1
 310 UNTILwin%=99ORturn%=42:PROCwin:GO
TO160
```

```
  320 END
  330 :
  340 ON ERROR OFF:MODE 6
  350 IF ERR<>17 THEN REPORT:PRINT" at
line ";ERL
  360 END
 1000 DEFPROCsetupgame
 1010 REM This draws the board, and est
ablishes who is playing.
 1020 VDU19,3,0;0;:MOVE285,895:DRAW285,
205:DRAW1020,205:DRAW1020,895:DRAW285,8
95
```


FOUR IN A ROW
The object is to get four counters of your own colour in a line in any direction.
You can choose to play against the computer or against another player. The machine always plays yellow.
Press the space bar to play.

```
 1030 MOVE300,880:MOVE300,220:PLOT85,10
05,880:PLOT85,1005,220:VDU20
 1040 VDU19,0,4,0,0,0
 1050 FORR=1TO6:FORC=1TO7:PRINTTAB(7+3*
C,2+3*R);C$;:NEXT:NEXT
 1060 VDU5:MOVE355,260:GCOL0,0:PRINT"1
 2  3  4  5  6  7":VDU4
 1070 VDU23,1,0;0;0;0;
 1080 REPEAT:PROCclear:PRINTTAB(11,27)"
Do you want to play":PRINTTAB(10,29)"th
e computer?(Y or N) ":ans$=GET$:UNTILan
s$="Y"ORans$="N"
 1090 IF ans$="N"THEN ENDPROC
 1100 REPEAT:PROCclear:comp%=1:PRINTTAB
(12,27)"Do you want to go":PRINTTAB(13,
29)"first?(Y or N) ":ans$=GET$:UNTILans
$="Y"ORans$="N"
 1110 IF ans$="Y" go%=0
 1120 ENDPROC
 1130 :
 1140 DEFPROCcolour
 1150 REM This colours display at next
available position
 1160 col%=column%:row%=0:REPEAT:row%=r
ow%+1:UNTILposition%(row%,column%)=0
 1170 position%(row%,column%)=1
 1180 COLOUR(130-go%):PRINTTAB(7+3*colu
mn%,23-3*row%);C$:COLOUR128
 1190 ENDPROC
 1200 :
 1210 DEFPROCcircles:REM Draws one circ
le
 1220 VDU23,224,255,255,255,252,240,224
,224,192:VDU23,227,192,128,128,128,128,
128,128,192:VDU23,229,3,1,1,1,1,1,1,3:V
DU23,226,255,255,255,63,31,15,7,3
 1230 VDU23,232,3,7,15,31,63,255,255,25
5:VDU23,228,0,0,0,0,0,0,0,0,0:VDU23,225,2
55,255,0,0,0,0,0,0:VDU23,231,0,0,0,0,0,
0,255,255
 1240 VDU23,230,192,224,224,240,252,255
,255,255
 1250 C1$=CHR$224+CHR$225+CHR$226:C2$=C
HR$227+CHR$228+CHR$229:C3$=CHR$230+CHR$
231+CHR$232:COLOUR3:COLOUR127
 1260 C$=C1$+CHR$8+CHR$8+CHR$8+CHR$10+C
2$+CHR$8+CHR$8+CHR$8+CHR$10+C3$
 1270 ENDPROC
 1280 :
 1290 DEFPROCtitles:VDU23,1,0;0;0;0;
 1300 VDU28,0,2,39,0:COLOUR 129:CLS:COL
OUR 3
 1310 PRINTTAB(11,1)"FOUR IN A ROW"
 1320 VDU26:COLOUR 128:COLOUR 6
 1330 PRINTTAB(2,5)"The object is to ge
t four counters"'"of your own colour in
 a line in any"'"direction."
 1340 PRINT'TAB(2)"You can  choose  to
play  against the":PRINT"computer or ag
ainst another player. The":PRINT"machin
e always plays yellow."
 1350 PRINTTAB(6,19)"Press the space ba
r to play."
 1360 REPEAT UNTIL GET=32
 1370 ENDPROC
 1380 :
```




```
 1390 DEFPROCwin:PROCclear:SOUND&11,2,1
00,121
 1400 REM This responds to a win or draw
```

```
1410 IF turn%=42 AND win%<>99 PRINTTAB
(12,27)"Honourable  draw!":GOTO1440
1420 IF go%=0 PRINTTAB(14,27)"Yellow w
ins!"ELSE PRINTTAB(16,27)"Red wins!"
1430 *FX15,1
1440 PRINTTAB(5,30)"Press space bar to
 play again.":IF GET$=" "THEN1450ELSECL
S:END
1450 ENDPROC
1460 :
1470 DEFPROCclear:PRINTTAB(0,27);SPC(1
20)
1480 ENDPROC
1490 :
1500 DEFPROCturn
1510 REM This displays whose turn next
, and accepts response
1520 turn%=turn%+1
1530 IF comp%=1AND go%=1THEN1610
1540 REPEAT:PROCclear:IF go%=0PRINTTAB
(8,27)"Reds turn - ";
1550 IF go%=1PRINTTAB(6,27)"Yellows tu
rn - ";
1560 *FX15,0
1570 PRINT"which column?":column%=VALG
ET$:UNTILcolumn%>0ANDcolumn%<8
1580 IFposition%(6,column%)>0THEN1540
1590 IF go%=1 go%=0 ELSE go%=1
1600 ENDPROC
1610 PROCclear:PRINTTAB(16,27)"Computi
ng"
1620 go%=0
1630 ENDPROC
1640 :
1650 DEFPROCcomputer
1660 REMThis must calculate a value fo
r column%
1670 flag%=0:IF turn%<4 column%=4:TIME
=0:REPEATUNTILTIME>100:ENDPROC
1680 dontgo1%=0:dontgo2%=0:dontgo3%=0:
dontgo4%=0
1690 swap%=0:column1%=0:column2%=0
1700 FORcol%=1TO7
1710 IFposition%(6,col%)>0THEN1790
1720 IF col%=dontgo1% OR col%=dontgo2%
 OR col%=dontgo3% OR col%=dontgo4% THEN
1790
1730 screen%=14347+48*col%:REPEAT:scre
en%=screen%+1920:UNTIL?(screen%+1920)>0
ORscreen%>26203
1740 IFscreen%>26203screen%=25867+48*c
ol%
1750 PROCweighandcheck
1760 IFweight%>swap%THENcolumn%=col%
1770 IFweight%>swap%THENswap%=weight%
1780 IFcheck%=3 col%=7
1790 NEXT
1800 IF column1%>0THENcolumn%=column1%
:ENDPROC
1810 IF column2%>0THENcolumn%=column2%
:ENDPROC
1820 flag%=flag%+1
1830 IF turn%<9 OR position%(5,column%
)=1 THENENDPROC
1840 screen%=14347+48*column%:col%=col
umn%
1850 REPEAT:screen%=screen%+1920:UNTIL
?(screen%+3840)>0ORscreen%>=22363
1860 PROCweighandcheck
1870 IF flag%>4 THENIF dontgo1%>0 AND
dontgo1%<>dontgo3% AND dontgo1%<>dontgo
4% THENcolumn%=dontgo1%:ENDPROC
1880 IF flag%>4 AND dontgo2%>0 THENcol
umn%=dontgo2%:ENDPROC
1890 IF flag%>4THENcolumn%=dontgo3%:EN
DPROC
1900 IF dontgo1%=column% OR dontgo2%=c
olumn% OR dontgo3%=column% OR dontgo4%=
column% THEN1690
1910 ENDPROC
1920 :
1930 DEFPROCweighandcheck
1940 REM This gives a weighting to eac
h possible move selected in PROCcompute
r, and also checks for a winning line a
fter each turn.
1950 weight%=0
1960 FORA%=0TO3
1970 B%=A%=0AND1ORA%>0AND38+A%
1980 FORC%=0TO3:check%=0
1990 FORD%=0TO3
2000 peek%=?(screen%+B%*48*(D%-C%))
2010 check%=peek%=240ANDcheck%+1ORpeek
%=15ANDcheck%-1ORpeek%=0ANDcheck%
2020 NEXT
2030 IFmissweigh%=0 THEN2090
2040 weight%=weight%-(75+RND(30))*(che
ck%=-2)-(15+RND(10))*(check%=-1)-(15+RN
D(10))*(check%=1)-(75+RND(30))*(check%=
2)
2050 IF check%=3 THENcolumn1%=col%:IF
dontgo1%=0 THENdontgo1%=col%
2060 IF check%=3 THENdontgo2%=col%
2070 IF check%=-3 THENcolumn2%=col%:IF
 dontgo3%=0 THENdontgo3%=col%
2080 IF check%=-3 THENdontgo4%=col%
2090 IF check%=4 OR check%=-4 win%=99
2100 NEXT:NEXT
2110 ENDPROC
```

ELBUG MAGAZINE CASSETTE

To save wear and tear on fingers and brain, we offer, each month, a casette of the programs featured in the latest edition of ELBUG. The first program on each tape is a menu program, detailing the tape's contents, and allowing the selection of individual programs. The tapes are produced to a high technical standard by the process used for the BEEBUGSOFT range of titles.

Magazine cassettes have been produced for each issue of ELBUG from Volume 1 Number 1 onwards and are all available from stock, priced £3.00 each inclusive of VAT. See below for ordering information.

This months cassette (Vol.1 No.6) includes:
Hunt The Numbers Game, Invisible Alarm Clock, Selective Renumber Utility for Basic programs, ASTAAD2 (The complete extended version of the original CAD program), Graphics Example Programs (3), Lunar Escape Game, Dancing Lines (an interesting visual display), and Four In A Row Game.

MAGAZINE CASSETTE - SPECIAL OFFER ON BACK ISSUES

The first 6 ELBUG magazine cassettes (Vol 1 No 1 - Vol 1 No 6) are offered to UK members for the price of £15 inclusive of VAT and post. This represents a saving of £5 (including post) and is only offered to members of ELBUG who place orders before 30th June 1984.

MAGAZINE CASSETTE SUBSCRIPTION

We are also able to offer ELBUG members subscription to the magazine cassette, this gives the added advantage of receiving the cassette at around the same time as the magazine each month. Subscriptions may either be for a period of 1 year or 6 months, however for an introductory period we are also offering a trial 3 months subscription. (NOTE Magazine cassettes are produced 10 times each year).

If required, subscriptions may be backdated as far as Volume 1 Number 1, so when applying please write to the address below quoting your membership number and the issue from which you would like your subscription to start.

AS A SPECIAL INTRODUCTORY OFFER, WE WILL ALLOW YOU TO CHOOSE ANY ONE OF THE FIRST 6 MAGAZINE CASSETTES ABSOLUTELY FREE, IF YOU SUBSCRIBE TO THE MAGAZINE CASSETTE FOR A PERIOD OF 1 YEAR, BEFORE JUNE 30th 1984.

MAGAZINE CASSETTE ORDERING INFORMATION

Individual ELBUG Magazine Cassettes £ 3.00
P & P: Please add 50p for the first and 30p for each subsequent cassette.
Overseas orders: Calculate the UK price including post, then deduct 15% VAT and add £1 per item.

Magazine Cassette Pack (Vol 1 No 1 - Vol 1 No 6) UK £15.00
Magazine Cassette Pack Overseas £19.00

Magazine Cassette Subscription

1 YEAR (10 issues)	£33.00 Incl.....	O'SEAS	£39.00	No VAT payable
6 MONTHS (5 issues)	£17.00 Incl.....	O'SEAS	£20.00	No VAT payable
3 MONTHS (3 issues)	£10.00 Incl.....	O'SEAS	£13.00	No VAT payable

Please be sure to specify that you require subscription to the ELBUG magazine cassette, and enclose your membersip number with a cheque made payable to BEEBUGSOFT.
Send to ELBUG Magazine Cassette, BEEBUGSOFT, PO Box 109, High Wycombe.

Subscriptions and Software Help Line
St Albans (0727) 60263
Manned Mon-Fri 1pm-4pm.

BACK ISSUES AND SUBSCRIPTIONS

BACK ISSUES (Members only)

All back issues will be kept in print (from November 1983). Send 90p per issue PLUS an A5 SAE to the subscriptions address. Back copies of BEEBUG are available to ELBUG members at this same price. This offer is for members only, so it is ESSENTIAL to quote your membership number with your order. Please note that the advertising supplements are not supplied with back issues.

Subscription and Software Address

ELBUG
PO BOX 109
High Wycombe
Bucks

SUBSCRIPTIONS

Send all applications for membership, and subscription queries to the subscriptions address.

MEMBERSHIP COSTS:

U.K.
£5.90 for 6 months (5 issues)
£9.90 for 1 year (10 issues)
Eire and Europe
Membership £16 for one year.
Middle East £19
Americas and Africa £21
Elsewhere £23
Payments in Sterling preferred.

SOFTWARE (Members only)

This is available from the software address.

MAGAZINE CONTRIBUTIONS AND TECHNICAL QUERIES

Please send all contributions and technical queries to the editorial address opposite. All contributions published in the magazine will be paid for at the rate of £25 per page.

We will also pay £10 for the best Hint or Tip that we publish, and £5 to the next best. Please send all editorial material to the editorial address opposite. If you require a reply it is essential to quote your membership number and enclose an SAE.

Editorial Address

ELBUG
PO Box 50
St Albans
Herts

ELBUG MAGAZINE is produced by BEEBUG Publications Ltd.
Editor: Mike Williams.
Production Editor: Phyllida Vanstone.
Technical Assistants David Fell, Nigel Harris and Alan Webster.
Managing Editor: Lee Calcraft.
Thanks are due to Sheridan Williams, and Adrian Calcraft for assistance with this issue.

All rights reserved. No part of this publication may be reproduced without prior written permission of the Publisher. The Publisher cannot accept any responsibility, whatsoever, for errors in articles, programs, or advertisements published. The opinions expressed on the pages of this journal are those of the authors and do not necessarily represent those of the Publisher, BEEBUG Publications Limited.
BEEBUG Publications LTD (c) 1984.

Printed in England by Staples Printers St Albans Limited at The Priory Press.